SHAWNEE MISSION PUBLIC SCHOOLS
SHAWNEE MISSION, KANSAS

JUNIOR HIGH COMPUTER LITERACY UNIT

EIGHTH GRADE MATHEMATICS

SEVENTH GRADE HONORS MATHEMATICS

1983-1984

Edited by:
> John Rezac, Mohawk Instructional Center

Developed and Written By:

> Jerry Belshe, Nallwood Junior High
> Cathi Duncan, Nallwood Junior High
> Robert Herdt, Nallwood Junior High
> Kristi Mehrer, Broadmoor Junior High
> Julie Porter, Broadmoor Junior High
> Barbara Turnbull, Broadmoor Junior High

Additional Assistance From:

> Connie Bonovich, Hocker Grove Junior High
> Tom Colley, Old Mission Junior High
> Gary Coulter, Nallwood Junior High
> Bobbie Leonard, Indian Hills Junior High
> Sharon Maxey, Nallwood Junior High
> Martha Tietze, Hillcrest Junior High
> Patti Whitaker, Meadowbrook Junior High
> Phyllis Zimmer, Milburn Junior High

The students will be able to:

1. Recognize and identify problem situations for which a computer can be used and develop strategies for their solutions.

2. Understand the tools and terminology of computers.

3. Maintain an interest in and a positive attitude toward computers.

4. Become aware of the impact of computers on their daily lives.

INDEX TO COMPUTER LITERACY STUDENT PACKET

WHAT ARE COMPUTERS?

Computers are machines that handle data, both facts and figures. Computers can solve problems, print words, draw pictures, store information, retrieve information, compare information, play games and do many other things. Computers should be simple and easy to use.

Despite what you may have heard, people control computers. People design computers, build them, sell them, and tell them what to do. People must write instructions for computers telling them what to do because computers can not think. Computers must be told everything - where to start, stop, print and what to do with a calculated answer. These written instructions are called computer programs. People who write the programs are called programmers.

Computers can perform three basic functions:

1) Calculate - add, subtract, multiply and divide

2) Compare - <, >, or =
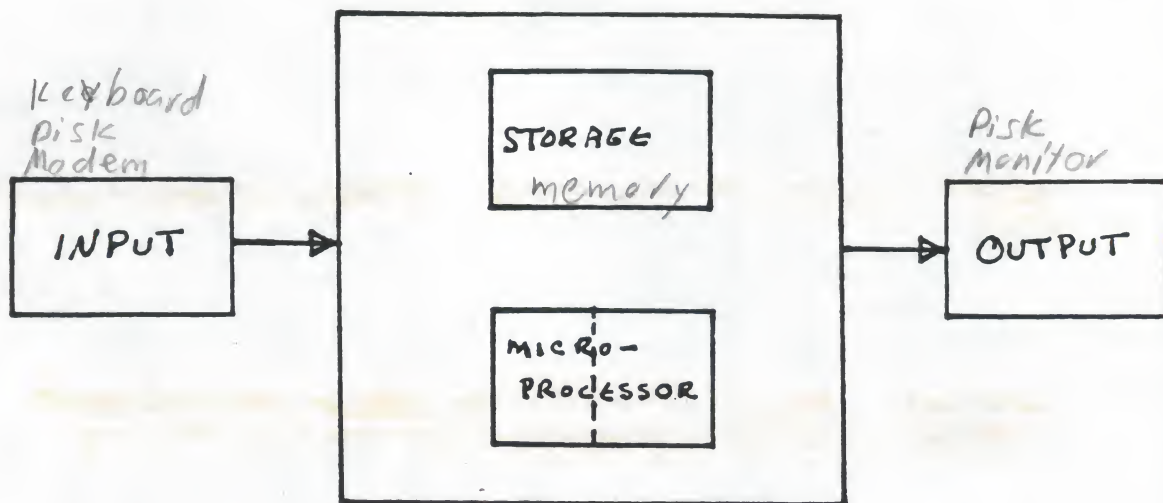
3) Store and retrieve data and information

Computers derive most of their amazing power from their features -

1) Speed

2) Accuracy

3) Memory

Computer System

The basic units or functions of the computer system we will be using are:

1) Input

2) Storage

3) Processing

4) Output

Keyboard
Disk
Modem

STORAGE
memory

MICRO-
PROCESSOR

Disk
Monitor

INPUT

OUTPUT

Input ——→ feeds instructions (computer programs) and data into the
computer system. Some input devices include the keyboard, the
diskette, the telephone, and a card reader.

Storage ——→ stores both data and instructions until needed. The storage of
the computer is called memory or main storage.

Processing ——→ interpreting (decoding) the instructions, controlling their
execution and performing all of the calculations.

The heart of the computer system is the Central Processing
Unit (CPU). The CPU consists of the following two sections:

1) Control unit - coordinates all activity much like a traffic
cop after a baseball or football game. It tells what part
of the machine needs to do what in order to accomplish the
parts of the program.

2) Arithmetic - Logical Unit - performs all calculations
and makes decisions based on the results.

Output ——→ provides answers and results (processed information) in any one
of several physical forms. Some of these forms include a
printed page, the information on the monitor, and the information
on a diskette.

THE EQUIPMENT

As you sit at your computer, you will see four parts of the computer system. These will become very familiar to you. As you read the descriptions below, look them over very carefully.

1) A monitor or CRT (Cathode Ray Tube). This looks like a television set but you cannot get any television channels on it. A television could be used, but the monitor gives a clearer picture for computer use. The monitor will show the words, numbers, and letters which you type into the computer. It will also show pictures, graphs, and diagrams. Monitors can be black and white, green, or color.

2) A computer. This looks like a low typewriter. The computer has two major parts. The first part is the typewriter keyboard. You will learn more about this later. The second part is the internal part of the computer. Inside are all of the components to do complicated operations. The operations will tell the monitor what to print out. It will run games, solve problems, draw pictures, make logical decisions, and store and retrieve information. You will learn more about the inside of a computer later. Never open the computer.

3) A disk drive. This is the small unit beside your computer. It is a mechanical device which will help you save your work or put information into the computer. There is a slot in the front and a small door which can be raised and lowered. Floppy diskettes can be put into the slot of the disk drive.

4) A Printer. At least one of the computers in your room will have a printer attached to it. This will allow you to print your answers on paper as well as on the monitor.

The four pieces of equipment listed above are called hardware. Hardware includes all of the equipment which is manufactured and which runs on electricity or batteries.

Besides hardware, a complete computer system requires software and the people who operate and use it. Software includes instructions and programs which tell the computer what to do. Software is frequently stored on a diskette. It is also called a floppy disk. The diskette is small and flexible and looks like a record inside of a sealed envelope. The inside part is called a diskette and is never to be removed from the sealed envelope. The sealed envelope protects the information stored on the inside part that rotates. Just as a record stores musical sounds, a diskette stores information so that it can be used over and over again. You are responsible for the diskette and need to return it in good condition. You also should never touch the parts of the diskette which you can see, as it is a very sensitive device. Similarly, never bend it, put books on it, or mishandle it in any other way.

People like to think of a computer as a device which solves problems. In order to solve problems, you must give it information, allow it time to solve the problem, and then expect it to give you some results. This is a three step process which takes place in this order:

1. Input
2. Processing
3. Output

This is sometimes written using arrows:

Input ⟶ Processing ⟶ Output

Input involves using the hardware devices which allow you to tell the computer something. The keyboard is an input device. You can type in some data or your directions. When you do that, your information is stored in the computer.

Once information is stored within the computer, the computer can start processing the data, programs, and information.

So that you can see what the answers are, the computer must show you the results somehow. The monitor and the printer are output devices. In one case, the computer puts the answer on a screen and, in the other case, it prints it on paper.

The diskette is both an input and an output device. You can read information from the diskette into the computer and you can also save information from the computer onto the diskette for later use. Many companies sell their programs by putting them on diskettes so that you can use their programs on your own computer.

THE COMPUTER KEYBOARD

The Apple Keyboard is very similar to a standard typewriter keyboard. The major differences are that there are a few extra keys and all of the letters are upper case. There are no lower case letters. The position on the monitor where you are typing the next character will be indicated by a flashing white square. This flashing white square is called cursor.

USING THE SHIFT KEY

Notice several keys have two symbols on them. If you press one of these keys, the lower symbol will appear on the monitor. If you press the same key while holding down one of the two SHIFT keys, then the upper symbol will appear on the monitor.

If there is no upper symbol on a key, then holding down the SHIFT while the key is pressed has no effect. There are two exceptions.

1. SHIFT M gives a right hand square bracket ( ]) although a bracket does not appear on the key.

2. The G key has the word BELL above. Holding down the SHIFT key and typing G will not give the BELL. It only gives you a G. (To get the BELL to ring use CTRL and G.)

USING SPECIAL KEYS

On a standard typewriter you can type the numeral "1" two different ways:

1. using the lower case "L", or
2. typing the numeral "1" (if your typewriter has it).

The Apple keyboard does not have lower case. The "1" and the "L" are two separate keys and the computer knows the difference between them.

Notice the difference between the letter O and the numeral zero. They look identical, but the computer can tell the difference. Notice that the numeral zero has a slash through it like this "Ø" so it does not look like the letter "O". To type a Ø you must use the Ø key and to type an O you must use the letter O key. The printer may not print the slash. This is because the printer has an oval zero and a square letter O.

The RETURN key is similar to the return key on an electric typewriter. This key causes the cursor to RETURN to the screen's left edge and must be pressed each time you wish to enter a line from the keyboard. By pushing the RETURN key, you are telling the computer that you have finished with that line.

CORRECTING SIMPLE ERRORS

Two other very important keys are the ← and the → keys. The ← key is called the backspace key. This key backspaces the cursor one space at a

time to the left. As the cursor moves, one character is erased from the program line which you are currently typing even though it does not disappear from the screen. The ← key, called the retype key, moves the cursor to the right. As it moves, each character it crosses over on the screen is entered just as though you typed it in from the keyboard.

These two keys are very useful in correcting errors. For example, let's suppose that you type

        PRINT "APPLE CIMPUTER"

before pressing the RETURN key. You can use the ← key and go back to the error. This would be the letter I. Now, retype the correct letter, which is O. To retype the rest of the line you could simply retype each of the characters MPUTER" or you could use the → key to retype each of the characters.

When corrected, the line will look like

        PRINT "APPLE COMPUTER"

The word computer is often misspelled by people who use it. It is spelled comput<u>er</u>, not comput<u>or</u>.

USING THE RETURN KEY

Everytime you type a line, you must push the RETURN key. This is standard procedure. It will become automatic. By pushing the RETURN key, you are telling the computer that you have finished with that line.

POWER SWITCH

The on/off switch is on the back of the computer. When it is turned on, the POWER light at the bottom left of the keyboard will light up. The POWER "key" is really not a key. It cannot be pressed down to turn on the system. Normally, your instructor will turn the computer on and off. You are <u>not</u> to use the switch unless instructed to do so.

The computer must always have a diskette in the disk drive when the computer is turned on.

USING THE CTRL KEY

Some keys have a third function which is activated by holding down the CTRL key and pressing a key. CTRL stands for control key. These special function keys will be explained as you need them. Three which are commonly used are:

        CTRL-C
        CTRL-S
        CTRL-RESET

APPLE II   KEYBOARD

PAGE 8
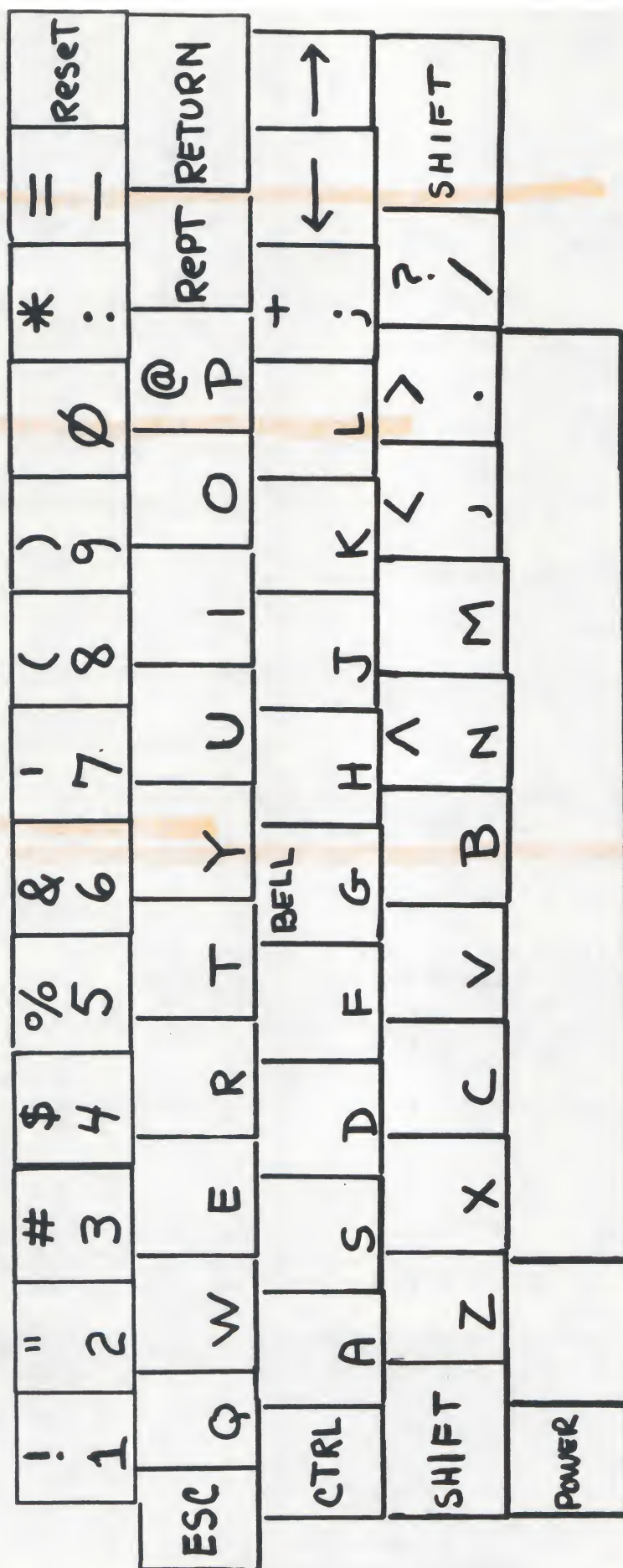
APPLE II KEYBOARD SUMMARY

1 and L	These are different keys.
Ø and O	These are different keys.
RETURN	Required at the end of each line
SHIFT	Pressed to get the upper symbol on those keys with two symbols
←	Backspace Key - characters are erased, but they do not disappear from the CRT.
→	Retype Key
REPT	Repeat Key - Used with other keys, you can type the same characters many times.
POWER	This is not a key. It only indicates when the computer is on.
CTRL	Control Key - This is another type of Shift Key. Control characters never appear on the monitor, but they cause the computer to do things.
RESET	Most computers have this option disconnected. CTRL-RESET will cause the computer to be "turned off" and then "turned on" again.
ESC	Escape Key - This is a third type of Shift Key which is usually used with other keys.
SPACE BAR	This long bar across the bottom of the Keyboard will insert a space and move the cursor over one space to the right.

IMMEDIATE MODE

The computer can operate in two different modes. After you learn to use the computer, you will find that you will use program mode in most situations. You may even forget that there is an immediate mode!! So exactly what is program mode and immediate mode?

You will learn about program mode later, but it allows you to save your steps and repeat the use of them. You can save your results on diskette. You can change your steps and cause a different problem to be solved.

On the other hand, immediate mode causes the computer to act like a calculator. Your results are immediate. You cannot save the steps or the problem. There are two commands which are most frequently used in immediate mode. One is the LET statement and the other is the PRINT statement. Other computer statements can be used as well, but they are used less frequently. The use of the others will not be discussed here.

CORRECTING ERRORS

You might make some errors when you use immediate mode. If you have already pushed the return key, then it is too late to correct the error. You must retype the line again.

If you have not pushed the return key yet, then you must backspace (the ← key) until the cursor (the blinking square) is over the mistake. Correct the mistake. Then either retype the rest of the line or use the retype key (the → key). You will find this very easy to do.

USING THE PRINT STATEMENT

Suppose that you want to know what the sum of 8 and 7 is. You simply type PRINT 8 + 7 and then press return. It will look exactly like this:

Example:

```
]PRINT 8 + 7
15
```

Notice that the answer appears immediately. The computer will give you the results when you use the word PRINT.

]Print 6+4
  10

]Print "6+4"
  6+4
]Print 5-2
  3
]Print 4*7
  28
]? 1/3 +2
  4

CL/Immediate/Page 1/10-82/a

Examples:

```
]PRINT 98 - 3
95

]PRINT 34*56
1904

]PRINT 789/4
197.25

]PRINT 8 ^ 3
512
```

The symbols which are used are:

+ addition
- subtraction
* multiplication
/ division
∧ raising to a power

You may do two or three of these on the same line.

Example:

```
]PRINT 8 + 33, 78 /3, 777*52
41                26              40404

]PRINT 777+3, 872+3456, 900*40
780               4328             36000
```

ORDER OF OPERATIONS

If the problem becomes very complicated, you may use one or more sets of parentheses. Since the computer needs to know which operation to do first, it automatically does them in the same order every time.

First, it does the results inside the parentheses.
Second, it raises all numbers to their powers.
Third, it does all multiplications and divisions.
Fourth, it does all additions and subtractions.

Look at the examples below. You will notice that the computer does the problems in the same order in which you have been doing it for years. It is the same order used in Algebra and arithmetic. This makes it very easy to remember.

Examples:

```
]PRINT 88+3*5
103

]PRINT 34*2 - 55*3
-97

]PRINT 72/3+67*3-30*4
105

]PRINT 4*(55+34)
356

]PRINT 66/(35-2) + 45*(3+2)
227
```

PRINTING WORDS AND NUMBERS

Sometimes, it is nice to print words. You may want to print TODAY IS MONDAY or THE ANSWER IS 84. These are both possible. Look at the difference between the use of the comma and the use of the semicolon. The comma causes the results to be separated farther than does the use of the semicolon. And last of all, notice what happens when you write PRINT all by itself. A blank line gets printed.

Examples:

```
]PRINT "TODAY IS MONDAY."
TODAY IS MONDAY.

]PRINT "THE ANSWER IS ",14*6
THE ANSWER IS    84

]PRINT "THE ANSWER IS ";14*6
THE ANSWER IS 84

]PRINT "THIS","IS","SOMETHING"
THIS            IS              SOMETHING

]PRINT "THIS";"IS";"SOMETHING"
THISISSOMETHING
.
]PRINT
```

]Print "Hello" Hello   ]? 456 | 4   5   6
]Print Hello 2.Syntax Error]?5;6;6|556
                         ]PRINT |

PAGE 12

USING THE LET STATEMENT

The LET statement allows you to save information, including numbers, within the computer so that you can use them later. The word LET is not required.

Example:

```
JLET A = 52

JLET B = 20

JC = 60

JPRINT A + B + C
132

JLET D = A + B + C

JPRINT D
132

JPRINT "THE SUM OF A, B, AND C IS ";D
THE SUM OF A, B, AND C IS 132
```

There are rules about the names of locations you can use, but they will be ignored here. The twenty-six letters of the alphabet are proper names for these locations. In the example above, A, B, and C were assigned the numbers 52, 20, and 60. When the first three lines were typed in, nothing appeared to happen. You will see that something really did happen to A, B, and C when the fourth line was typed in the computer. The computer knew what A + B + C was. Check the next two lines. D became the value of what A + B + C is. Then, the value of D was written out. And finally, the last line should be looked at carefully. A message is written out and a value of what is stored in a location is written out. The last line has more meaning to somebody than simply an answer of 132. You should always give some thought about the person who will read the results instead of the person who wrote the problem.

SAVING AND PRINTING MESSAGES

Numeric locations are used to store numbers and string locations are used to store messages. String variables must have a $ attached at the end of the name. A$, B$, and C$ are string variables. This allows messages to be saved. Look at the examples below. You may change what is stored in a location. This is done with the variable NAME$. Exactly the same thing can be done with numeric variables. You can store different things in the same location at different times. When you replace a location with something new, the old value just disappears.

Examples of numeric variables

$C = 16 \quad A = 5 \quad B = 2$

strings variables

$A\$ = "Jim" \quad B\$ = "cat" \quad C\$ = "school"$

        ]SCHOOL$ = "SHAWNEE MISSION "

        ]LET NAME$ = "NORTHWEST"

        ]PRINT SCHOOL$; NAME$
        SHAWNEE MISSION NORTHWEST

        ]LET NAME$ = "SOUTH"

        ]PRINT SCHOOL$;NAME$
        SHAWNEE MISSION SOUTH

USING THE COMPUTER AS A LIE DETECTOR

Have you ever wondered if 8 = 3? The computer could tell you.

Example:

        ]PRINT 8 = 3
        0

The computer looks to see if 8 equals 3. If they are equal, that is, the statement is true, then a 1 is typed. When the statement is false, a 0 is typed. There are a total of six symbols which can be used. These are:

SYMBOL	MEANING
=	EQUALS
<	LESS THAN
>	GREATER THAN
<=	LESS THAN OR EQUAL TO
>=	GREATER THAN OR EQUAL TO
<>	IS NOT EQUAL TO

Examples:

        ]PRINT 18 + 21 <20
        0

        ]PRINT 8 + 8 >=16
        1

        ]PRINT 45 - 34 <> 5 * 3
        1

USING AND, OR, AND NOT

We can combine some of these together to see if they are true or false by

PAGE 14

using AND, OR, and NOT.

Examples:

```
]PRINT 3 =2 + 1 AND 5 = 3 + 2
1

]PRINT  6 >2 *2 AND 5 < 3 * 1
0

]PRINT NOT  5 = 4 + 1
0

]PRINT 1 = 1 OR 2 = 3 OR 4 = 5
1
```

Here are the rules for AND, OR, and NOT.

AND   When used, every part must be true in order for
        the statement to be true. If at least one part
        is false, then the entire statement is false.

OR   When used, at least one part must be true in order
        for the statement to be true. If all parts are false,
        then the statement is false.

NOT   When used, this makes the statement the opposite of
        what it normally would be. It makes a true statement false
        and a false statement true.

Remember that 1 indicates that the entire statement is true and 0
indicates that the entire statement is false.

THE USE OF THE = SYMBOL

The equal sign is used two ways with the computer and both are different
from your use of it in mathematics.

When you use it in a LET statement, it means "stores the value of." In
the example, X stores the value of 82.

Example:

```
]LET X = 82

]PRINT X
82
```

When you use a logical equals sign, the computer tells you if it is true
or false. In the example, the computer tells you if X "is equal to" 82 or

not. The value of X was saved from the previous example, so the answer is true.

Example:

```
]PRINT X = 82
1
```

Now it is your turn to practice using the computer as a <u>very expensive</u> <u>calculator</u>. This is called using the computer in immediate mode.

VARIABLES

Variables are the names of locations in the computer where information is stored. It may be helpful to think of a variable name as a name which is assigned to a letter box. Anything placed in the letter box becomes the value associated with the letter box name, until something new is placed in the letter box. We say a value is stored in a variable.

The real power of a variable is that it does not always have to have the same value. Variables can change in value.

Example: Type the following and notice what happens.

```
LET X = 5                    ] Run
PRINT X
LET S = 3                    5
PRINT S                      3
PRINT X + S
LET X = 10                   15
PRINT X                      10
PRINT X + S                  13
```

What happened to the value of X in this sequence?

You have used variables in math before. In math classes, variables were represented by a square, a triangle, a blank line, or a letter of the alphabet. Math variables almost always represent one or several answers, and you are required to find the correct one. Computer variables are simply locations in the computer where you can put things. You can put lots of things in those locations. You can put different numbers at different times. You can even put words and sentences in the locations. Of course, you can also put numbers. Variables are temporary boxes where you can store things to help you solve your problems. Computer variables are different from math variables.

RULES FOR VARIABLES

You must follow certain rules for naming variables. They are easy to remember.
You should know that different computers and different languages use different
rules for naming variables. These are the rules for BASIC on the Apple II micro-
computer.

Variables are indicated by letters or a combination of letters and numbers. The
four rules which must be followed are:

1. Variables **must** start with a letter of the alphabet.

2. Variables must contain only letters and numbers and no special
    character such as "#" or "," or "+".

3. The computer only looks at the first two characters. Thus, AP
    and APF are the same variable locations.

4. There are some reserved words which the computer understands,
    such as HOME and LET and PRINT. These words, as well as words
    containing them, cannot be used. *Refer to page 108*

TYPES OF VARIABLES

Variables can be of two types. They can be numeric or string.

A. A numeric variable is a location which can store only numbers.
    The numbers may be positive, negative, or zero. Since it is a
    physical location, there is a maximum and a minimum number which
    will fit in the space available. The four rules above tell you
    how to name a numeric variable.

B. A string variable is a location which can store numbers, letters,
    and special symbols. There are a maximum number of characters
    which can be stored in string variable locations. The rules are
    exactly the same as the numeric variables except that a $ is placed
    at the end of the variable name. You cannot do arithmetic with
    string variables.

Examples:

LEGAL NUMERIC VARIABLES	LEGAL STRING VARIABLES	ILLEGAL VARIABLES	*Should be*
A	A$	$A	*A $*
AB	AB$	A$B	*AB $*
CUF	CUF$	D3FOR	*P3*
NAME	NAME$	MATH$	*MH $*
A4	A4$	$	*(eHc-)$*
VOLUME	PART$	HISTORY	*His y*
AREA	ZIP$	23C	*C23*

The words "FOR", "AT", AND "OR" are three reserved words which appear in some of the illegal variables listed above. These are sometimes tricky to spot. There are about 100 reserved words. These are included on the Reserved Words List.

Variables should be meaningful names, and not just A, B and C. They should, however, be short.

EDITING PROGRAMS

Everyone makes mistakes. Sometimes the computer even makes mistakes because we type in the wrong numbers or we push the wrong buttons. We want to talk about your mistakes and how you can correct your mistakes. This process is called editing. When you correct programs, you are editing programs or debugging programs. In early computers, occasionally insects, or bugs, had to be removed from the computers before they would work correctly. Those people who removed the early insects from the computers called the process "debugging a computer." We now talk about debugging computer programs.

BEFORE YOU PUSH RETURN

Frequently, you will start typing and then realize that you have misspelled something. This is an easy mistake to correct. Use the backspace key (←) until the cursor is on the mistake. You then retype the correct character and retype the rest of the line again.

When you use the backspace key, the characters you backspace over do not disappear. However, they have been erased. This means you must either retype them or use the retype key (→). A little practice will make this very easy!
Example: Suppose that you have typed:

    ]100 PRINT "I NOVER MAKE MISTAKES."

When you realize that you did make a mistake and you didn't push the return key yet, you simply do these steps.

1. Backspace to the O
2. Retype an E
3. Use the retype key until the cursor reaches the end of the line
4. Push the RETURN key

Now it looks like this.

    ]100 PRINT "I NEVER MAKE MISTAKES."

AFTER YOU PUSH RETURN

Sometimes you have no other choice but to retype your information when you make an error. Notice in the example that both LIST and RUN were both misspelled. A "?SYNTAX ERROR" was printed and the bell sounded when the error was printed out. Your only choice is to retype the commands correctly.

Example:
    ]LOST

    ?SYNTAX ERROR
    ]RUUN

    ?SYNTAX ERROR

PAGE 20

As you type programs into the computer, you must use line numbers. Using line numbers correctly will be the secret to easy editing of your program. Use line numbers which are multiples of ten so that there will be extra space when you need it.

Correcting programs is a frequent activity. Corrections tend to fall into three categories.

1) Deleting a line -- type the line number and return. This will remove the line that was in the program.

2) Adding a line -- determine where in your program you want your new line. Pick a line number between the two line numbers where you want to insert the new line. Type in your new line, using the line number.

3) Changing a line -- determine the line you want changed. Type in a new line, using the same line number. This will erase the old line and replace it with the line you have just typed in.

In the example below, look at the original program. Line 40 needs to be changed. Lines 50 and 130 need to be deleted and line 150 needs to be inserted.

Example: This is the original program.

```
JLIST

10   REM   -- PROGRAM #1
20   REM   -- WRITTEN BY JIM ANDREWS
30   REM   --      AND MARY SMITHSON
40   REM   -- DUE ON 10/12/1964
50   PRINT "THE ANSWER IS ",SUM
100  LET BOYS = 13
110  LET GIRLS = 14
120  LET SUM = BOYS + GIRLS
130  LET GIRLS + BOYS = SUM
140  PRINT
999  END
```

PAGE 21

These are the corrections.

```
]40 REM -- DUE ON 10/12/1984

]50

]130

]150 PRINT "THE ANSWER IS ",SUM
```

This is the corrected program.

```
]LIST

10   REM  -- PROGRAM #1
20   REM  -- WRITTEN BY JIM ANDREWS
30   REM  --       AND MARY SMITHSON
40   REM  -- DUE ON 10/12/1984
100  LET BOYS = 13
110  LET GIRLS = 14
120  LET SUM = BOYS + GIRLS
140  PRINT
150  PRINT "THE ANSWER IS ",SUM
999  END
```

There are other editing techniques using the CTRL-X key, the Escape key, and the DEL command. You may want to explore these, but you will not be given information about them here.

PAGE 22

A system command has no line number (such as 10 PRINT "HELLO") and is not a part of a BASIC program. It is an instruction to the computer to do a specific task at the time the command is issued. There are many system commands. This paper reviews three system commands which you will use everytime you use a computer. They are the NEW, LIST, AND RUN commands.

USING THE NEW COMMAND

The NEW command clears the computer's memory of any previous programs. It is a good idea to type NEW before you begin a program to make sure the memory is clear. This command will erase any programs you may have had in the computer. If you do not type NEW before you begin a new program, then your old program and your new program will be all mixed up.

USING THE LIST COMMAND

The LIST command will print a list of all program statements (all statements with a line number). The LIST will always list statements in ascending (going from least to greatest) line numbers, regardless of the order in which they were typed.

Example: When you type in these lines,

```
100 LET X = 5
110 PRINT X
120 PRINT
115 PRINT X * 2
999 END
LIST
```

the LIST command will print it out like this.

```
100   LET X = 5
110   PRINT X
115   PRINT X * 2
120   PRINT
999   END
```

Notice the order of the line numbers!

When you list your program and see that you have made an error, you may correct it by simply retyping the entire line.

Example: When you type in these lines,

```
100   LET X = 2
110   PRINT "X + 2 = ";X + 2
120   PRINT
130 PRENT"X + 4 = ";X + 4
999   END
LIST
```

PAGE 23

the computer will display it in this form.

```
100   LET X = 2
110   PRINT "X + 2 = ";X + 2
120   PRINT
130 PRENT"X + 4 = ";X + 4
999   END
```

You notice the error in line 130, so you type

130 PRINT "X + 4 = ";X+4

LIST

and the entire program is listed with line 130 corrected.

You may just want to list one line in the program. If so, then you can type

LIST 110

and line 110 would be printed.

Example:

LIST110

110   PRINT "X + 2 = ";X + 2

If you want to list lines 100 through 120, type

LIST 100, 120

This will cause all lines from 100 to 120 (including 100 and 120) to be displayed.

Example:

LIST 100,120

```
100   LET X = 2
110   PRINT "X + 2 = ";X + 2
120   PRINT
```

In summary, most of the programs you write will be short. You will find it convenient to LIST the entire program by simply typing

LIST

On the other hand, you may choose to LIST one line or several lines close together by typing one of the following:

LIST 110

LIST 100,110

PAGE 24

DELETING A LINE

It sometimes happens that you typed in a line by mistake. Perhaps you
do not want to change the line, you simply want to delete it. To delete
a line, you can type in the line number and push the RETURN key. This
will erase that line from the memory of the computer. To delete several
lines, push the RETURN key after you type in each line number which you
want deleted.

Example: Assume that the following is typed in.

```
100   PRINT "COMPUTER LITERACY"
110   PRINT "FOR"
120   PRINT "JUNIOR HIGH STUDENTS"
130   PRINT "AND"
140   PRINT "ALL GRADE LEVELS"
150   END
```

The following shows how to delete lines 120 and 130 and then
LIST the new version of the program.

```
120
130
LIST
```

```
100   PRINT "COMPUTER LITERACY"
110   PRINT "FOR"
140   PRINT "ALL GRADE LEVELS"
150   END
```

You will notice that the two lines no longer appear in the
program.

USING THE RUN COMMAND

After you have listed the program and are reasonably certain that every-
thing has been typed correctly, you can cause it to be executed (the
computer will perform each of your commands) by using the command RUN.
(You do not have to LIST a program before RUNning it, but it is usually
a good idea.)

Example: When the following is typed in
```
100   LET X = 2
110   PRINT "X + 2 = ";X + 2
120   PRINT
130   PRINT "X + 4 = ";X + 4
999   END
```
RUN

these results will be displayed

X + 2 = 4

X + 4 = 6

Notice the difference between the commands RUN and LIST.

Computer Literacy

SYSTEM COMMANDS - PART II

You have already used a group of system commands which do not require the use of the disk drive. The commands were LIST, RUN, and NEW.

We will now refer to a variety of system commands which use the disk drive. Before we talk about the commands, we need to mention exactly how the disk drive is used with the computer.

PUTTING THE DISKETTE IN THE DRIVE

There is only one correct way to put the diskette into the drive. Someone created "the rule of the right thumb" so that it would be easy to remember. Grasp the label on the diskette with your right thumb. The label will be up. Then gently slide it into the disk drive. Your diskette should now be correctly positioned in the disk drive.

THE DISKETTE

The diskette which you will use can store programs. It can store your programs, your friend's programs, or programs which you may have purchased. Programs are a set of instructions which will tell the computer how to solve a particular problem. A program may be a game, a business payroll, an inventory system, a math problem, a letter-writing program, or a simulation. In order for the computer to use the program, the computer must know that the disk drive is connected to the computer. It must also get a copy of the program from the diskette and place it into the computer.

There are two ways to tell the computer that there is a disk drive connected. This process of telling the computer is called "booting the disk." The first method is to turn on the computer with a diskette in the drive. The light on the drive will automatically go, run for 10 to 30 seconds, and then turn off. The computer can now access the disk. The second method is to type

```
PR#6        OR
IN#6
```

Exactly the same process will take place. In either case, any program which was in the machine will now be erased. The disk drive is connected to the computer and the memory of the computer is now blank.

NEVER remove a diskette while the disk drive's "IN USE" light is on. This may permanently damage the diskette and the information on it. Should the drive need to be turned off, open the lid of the drive, push CTRL-RESET for about 10 seconds. The drive will stop, but you may have lost a program.

COMPUTER MEMORY AND DISKETTE MEMORY

The memory in the computer is of two different types. These are frequently called ROM and RAM.

ROM (Read Only Memory) is a type of memory which is random access. This means that it is available to the computer all the time. Permanently stored information is in ROM and it is put in the computer at the place where it is manufactured. You cannot change the information stored in ROM without replacing part of the computer. The computer uses it often and you have the use of it whenever you need it.

RAM (Random Access Memory) is a type of memory which is also random access. However, RAM is eraseable and can be used by you or the computer for a variety of uses. Your programs are stored in RAM when you make a copy of them from the diskette to the computer, or when you type in a program from the keyboard into the computer. When the computer is turned off or when you type a new program into the computer or when you copy a program from the diskette to the computer, the old program is erased. This also happens when you type the system command NEW.

RAM is called main storage, primary storage, main memory, primary memory, or simply "memory". The only program which the computer can run is the program in primary memory. Diskette memory is called secondary memory or secondary storage. Diskette memory is a convenient place to temporarily or permanently store a computer program. You can purchase programs on diskettes. You can also store your own computer program on a diskette so that you can use it again at a later time. In fact, you can use it as often as you want to and on whichever Apple II computer you want to.

The difference between RAM memory and diskette memory is very important. RAM memory is eraseable and reuseable. It is inside the computer. It is where the program must be in order for the computer to run the program. Diskette memory is also eraseable and reuseable. It is on the diskette and not in the computer. It is a storage which can be moved from one computer to another. It also allows you to save the programs you type in the computer.

We need to talk about the group of system commands which move copies of the programs from the computer to the diskette and from the diskette to the computer.

TO LOAD, RUN, AND SAVE A PROGRAM

To LOAD a program called SAMPLE, you would type the following

    LOAD SAMPLE

You now have two copies of the program - one on the diskette and one in the computer. You must know the name of the program which you wish to copy from the diskette to the computer. Names of programs must start with one of the 26 letters of the alphabet. Any characters are allowed after the letter of the alphabet -- letters, numbers, or special characters. Names of programs should be relatively short and describe what the program is about. Some good names are PROGRAM 1, PROBLEM - 14A, GRAPH, DUE FRI 10/22, etc. Program names like GEROGE, WINNER, XYZ, A3, FIGHT, SOMETHING, etc. are correct, but not

usually very useful. 8FOR, :XYZ, and 63 are invalid names for programs.

The LOAD command must include the name of the program. A copy of the program will be made from the diskette into the computer. The diskette will not be changed. The old program in RAM in the computer will be erased when the new one is copied.

Some examples are:
    LOAD DUE FRI 10/22
    LOAD GRAPH
    LOAD PROBLEM-14A

You may now RUN the program. RUN runs the program currently in the computer. There is a command which does these two steps at once. Type

    RUN SAMPLE

This is the same as if you were to type

    LOAD SAMPLE
    RUN

Using the RUN command with the name of the program after it causes the following to happen. First, the computer looks for the name of the program on the diskette. If it does not find it, it stops. If it does find it, it makes a copy of it into main memory. (The old program gets erased.) It then proceeds to RUN the program.

To write this in outline form, the results would look like this.

1. Computer tries to find SAMPLE on the diskette.

2. If SAMPLE is not on the diskette THEN go to step 5 below.

3. A copy of SAMPLE is placed in primary memory (Comment: the old program is erased.)

4. SAMPLE is run by the computer.

5. The computer returns control to the user.

Another way to show this process is to use a flowchart. Notice that what was said in written form was also written in outline form. The flowchart shows it in a pictorial form.

PAGE 28
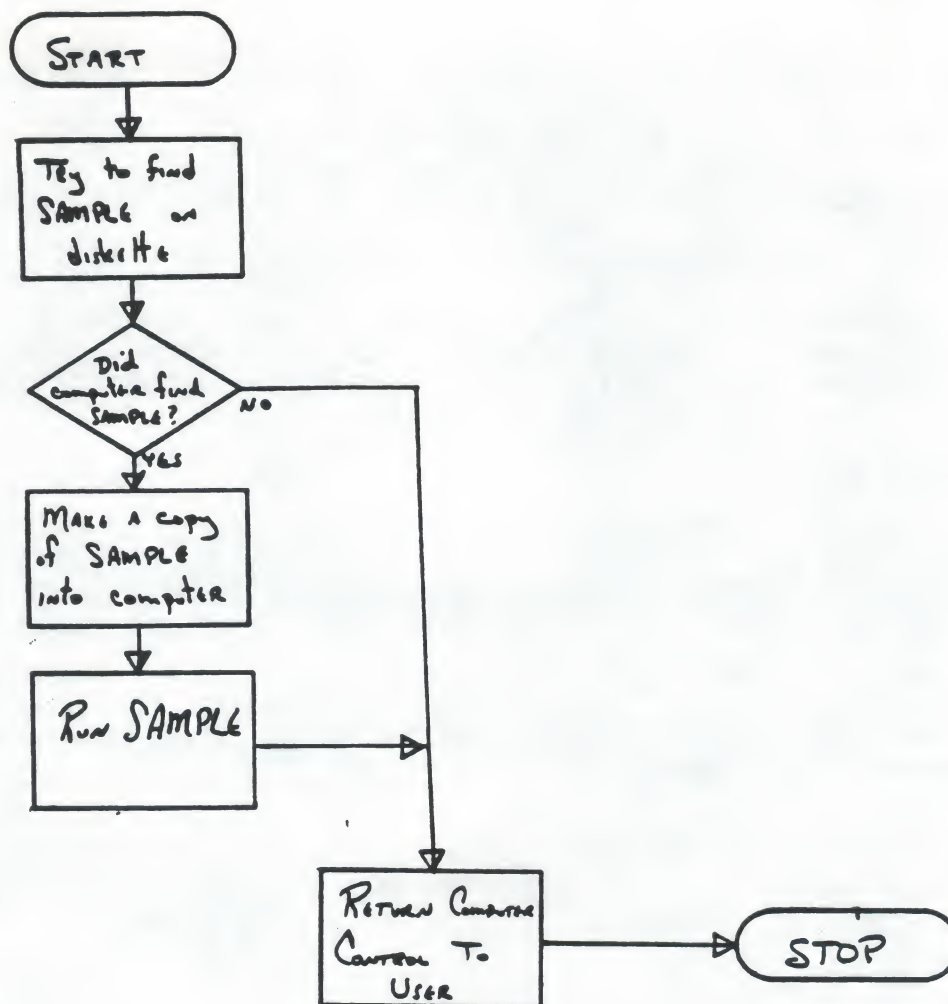
The SAVE command looks very much like the LOAD command. You can only save on the diskette the program which is currently in the memory of the computer. You __must__ give it a name. The name must follow the same rules listed above. Programs which are in the computer do not have names. If you want it to save the present program on diskette, and call it SAMPLE2 on the diskette, you would type

    SAVE SAMPLE2

You now have two copies of the program - one in the computer and one on the diskette.

If you already had a program called SAMPLE2, your old program on the diskette is now replaced with a later version. You no longer have the older version. If you did not have a program called SAMPLE2 on the diskette, then space has been allocated to save it under the name SAMPLE2. Look at the flowchart for this under the LOCK command later in this handout. An outline looks like this.

1. The computer tries to find SAMPLE2 on the diskette

2. IF SAMPLE2 exists on the diskette
    THEN replace SAMPLE2 with new version
    ELSE save SAMPLE2 in a new space on diskette

3. The computer returns control to the user.

In summary, LOAD PROB1 makes a copy of a program from the diskette to the RAM of the computer. SAVE PROB2 makes a copy of a program from the computer to the diskette. RUN PROB3 loads a program from the diskette to the computer and immediately starts the program running.

TO DELETE A PROGRAM

When you want to erase a program from the memory of the computer, you type NEW. This does not affect the diskette. To erase a program from a diskette, you must know the name of the program. If the name of the program which you want to delete is called PROJECT8, then

    DELETE PROJECT8

will erase the program entirely from the diskette. Nothing happens to the program within the computer.

TO CATALOG, LOCK, AND UNLOCK PROGRAMS

Suppose that you have forgotten the name of your programs on your diskette. Now, what do you do? That's easy! Simply type

    CATALOG

and the names of all of the programs on the diskette will be listed on the monitor. When you type CATALOG, you may see an asterisk in front of some of the names of the programs. This means that the program is locked. For example,

    ]CATALOG

    DISK VOLUME 254   = locked

    *A 002 HELLO
     A 002 SAMPLE2
     A 002 SAMPLE 2
     A 002 SAMPLE3
    *A 002 SAMPLE4

PAGE 30

First, notice that SAMPLE2 and SAMPLE 2 are two different programs. They
are spelled differently, so they are different!! The HELLO program should
never be deleted from your diskette. It was placed there by your instructor.

Also, notice that SAMPLE4 has an asterisk in front of the name. This
means that SAMPLE4 is locked. It cannot be erased or deleted by accident.
You can LOCK programs by typing

> LOCK SAMPLE4

And you can UNLOCK programs by typing

> UNLOCK SAMPLE4

This allows you to protect your own programs from your own errors. Now,
let's look back at the SAVE command. What happens if you type

> SAVE SAMPLE2

and SAMPLE 2 is locked? Read the flowchart carefully below so that you
understand the relationship between SAVE, LOCK, and UNLOCK.
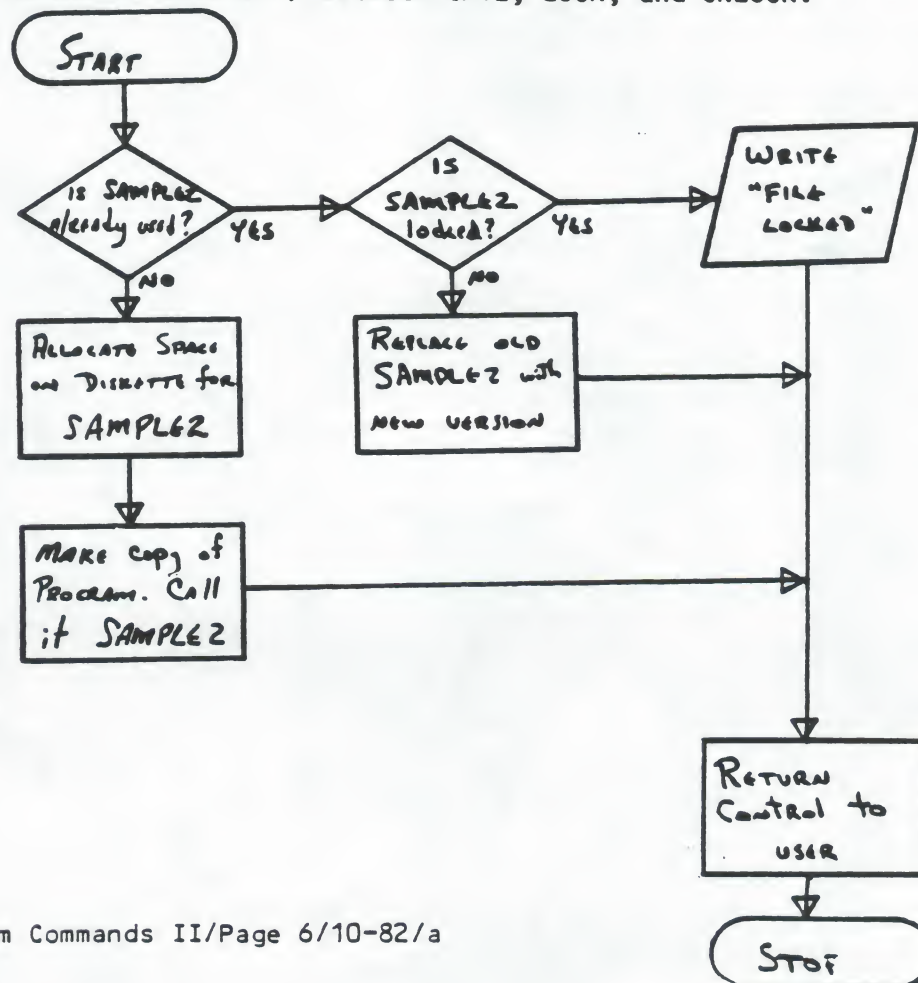
PAGE 31

USING THE PRINTER

There is one last command and then you can re-read these pages to be sure that you understand how you use your diskette.

The Command PR#1 tells the computer to write everything to the printer. The Command PR#0 says to the computer to write everything on the monitor.

After you have saved your program on a diskette and called it PROJECT7, you will probably want to have a printer write it out. Take your diskette to a computer with a printer. Turn on the printer by typing

PR#1

Load and run your program by typing

RUN PROJECT7

and then

LIST

Turn the printer off by typing

PR#0

and erase the memory (in the computer) by typing

NEW

You have now just made a copy of the running and listing of PROJECT7 on paper. You should understand the need for each of the steps above. If you did not, re-read these pages again and check with your instructor.

STARTING TO PROGRAM

The most useful mode of the computer is Program mode. In Program mode, you can write in your steps, save your steps, and execute them when you want, as often as you want. Program mode is indicated by using line numbers in front of each statement. The computer automatically puts the lines in order from smallest to largest, regardless of the order in which you enter them. A sample program follows. You have the option of LISTing the program or RUNning it. Both are shown.

```
JLIST

10   REM
20   REM   -- SAMPLE PROGRAM 1
30   REM
100  LET X = 4
110  LET Y = 12
120  LET Z = X + Y
130  PRINT "THE SUM IS ",Z
999  END

JRUN
THE SUM IS          16
```

In this study guide, the REM, LET, PRINT, and END statements will be described. When you have read these pages, you should be able to create simple programs using these four statements. Now, let's look at this program in some detail.

There are three LET statements. These statements, lines 100-120, put numbers into the locations called X, Y, and Z. The number 4 is placed in X. The number 12 is placed in Y. The number that is placed in Z is determined by adding the two numbers already in locations X and Y. The number 16 is placed in location Z.

The REM statements, lines 10-30, are comments which help you to jot helpful notes in the beginning or middle of your program. REM stands for the word "remarks". You can look at the word REMark and see why REM is used.

The END statement is required in all programs and must be the last (and highest numbered) statement. This tells the computer that there are no more statements to be executed. Try to use 999 or 9999 for the END statement and you will find that it will become a habit.

The PRINT statement, line 130, writes out a message ("THE SUM IS"), exactly like it is printed in the quotation marks, followed by the number stored in the variable location called Z.

The listing and the execution of the second sample program follows:

PAGE 33

```
JLIST

10   REM
20   REM  -- SAMPLE PROGRAM 2
30   REM
100   LET MILES = 160
110   LET TIME = 3
120   LET MPH = MILES / TIME
130   PRINT "THE MILES DRIVEN WAS ";MILES
140   PRINT "THE TIME NEEDED WAS ";TIME
150   PRINT
160   PRINT "THE RATE (IN MPH) WAS ";MPH
999   END

JRUN
THE MILES DRIVEN WAS 160
THE TIME NEEDED WAS 3

THE RATE (IN MPH) WAS 53.3333333
```

NOTE again the use of the REM and END statements. These have not changed much in the second example.

The variable MILES is assigned the value of 160 and the variable TIME is assigned the value of 3. The variable MPH is assigned the value of 160 divided by 3. This happens because the numbers in MILES and TIME are used to determine the value of MPH.

Four PRINT statements are used in this program. Lines 130, 140, and 160 print out both a message and a number in a variable location. Line 150 doesn't have anything after the word PRINT. This causes the computer to print out a blank line.

SAMPLE PROGRAM 2 is longer than SAMPLE PROGRAM 1. The results of 2 are more meaningful. The variable names of 2 are meaningful. The spacing of the answers in 2 are better.

We will now list the rules for each of the four BASIC statements used in the two sample programs.

REM STATEMENTS

REM stands for remarks. REM statements are used primarily for documentation. There are four major uses of REM statements.

1. You should include at the beginning of each of your programs your
    name, the program number, the date due, and any other information

PAGE 34

which your instructor requires. This allows you to identify each program. You should add any comments you want to remember about the program. Programmers usually include a complete description of the program and the definition of variables. This form of documentation is always included at the beginning of the program.

2. It is often desirable to include blank spaces in a program for easy readability. SAMPLE PROGRAM 2 contains blank spaces in lines 10 and 30. Sometimes as many as half of the REM statements in a program are used for this purpose. It makes the program neat and professional.

3. When a group of statements in a program serve a purpose, it is helpful to document or describe that purpose immediately before those statements. This will become clear as you write more programs.

4. It is desirable when the computer makes decisions that those decisions end at the same place. This will also become clear when you work with Choices and Loops.

    The REM statement requires a line number, the word REM, and then anything you wish. The REM is not outlined or flowcharted. The example below illustrates only the first two kinds of documentation. If you were to RUN this program, you would see that nothing is printed and nothing happens.

Example:

```
]LIST

10   REM
15   REM   -- WRITTEN BY RONALD REAGAN
20   REM   -- PROGRAM 123
25   REM   -- DUE ON FRIDAY, MARCH 6
27   REM   -- NOTE: FLOWCHART REQUIRED!!
30   REM
33   REM
35   REM   -- THIS PROGRAM DETERMINES THE SUM
40   REM   -- AND AVERAGE OF MY TEST SCORES
45   REM
```

LET STATEMENTS

    The LET statement has been discussed earlier, so only the highlights will be mentioned here. The line number is required to use a LET statement in a program. The word LET is optional. A single variable is required and the equal sign follows it. The right side of the equal sign can be very simple or very complicated.

PAGE 35

The equal sign means "is replaced by." The thing on the right
side of the equal sign is evaluated and the result is placed in the
variable location listed on the left of the equal sign.

Example:

```
]LIST

100   REM
110   REM  -- NUMERIC VARIABLE LET STATEMENTS
120   REM
130   LET A = 5
140   LET B = 10 * 32
150   LET C = A - B
160   LET D = A
170   LET E = A + B / (C - D)
200   REM
210   REM  -- STRING VARIABLE LET STATEMENTS
220   REM
230   LET A$ = "MATHEMATICS"
240   LET B$ = A$
300   REM
310   REM  -- WORD "LET" IS NOT USED
320   REM
330 Z = 45
340 Z$ = "END OF EXAMPLE"
```

PRINT STATEMENTS

The PRINT statement has several options. You can print a
message or a variable. You can print several messages and
variables. Spacing can be done wth either commas or semicolons.
Commas space the answers in columns. Semicolons place one item next
to the other without any spaces between them. You can mix commas
and semicolons on the same PRINT. The PRINT can also be used so as
to cause a blank line to be printed.

When a comma or a semicolon is the last character on a PRINT
statement, the computer does not go to the next line immediately.

Examples:]LIST

```
100   REM
110   REM  -- EXAMPLES OF PRINT STATEMENTS
120   REM
130   PRINT A
140   PRINT A,B,C
150   PRINT A;B;C
160   PRINT A,B;C,D;E;F
170   PRINT A$,B$
180   PRINT "THIS IS A MESSAGE."
190   PRINT "THE ANSWER IS ",A
200   PRINT "THE ANSWER IS ";A
210   PRINT "A = ";A,"B = ";B,"C = ";C
220   PRINT
230   PRINT A,B,C,
```

PAGE 36

The two examples below give the same answers, but the PRINT's
look different. In the first, the value of C is written out. In
the second, the computation is done so part of the PRINT and the
result is written.

Examples:

```
]LIST                          ]LIST

100   LET C = 4 + 5 - 3        100   PRINT 4 + 5 - 3
110   PRINT C                  999   END
999   END
                               ]RUN
]RUN                           6
6
```

END STATEMENTS:

Very little needs to be said about the END statement. There
can be only one END statement per program. It must be the highest
numbered statement in th program. It is suggested that you always
use 999 or 9999 for your END statement.

PAGE 37

PATTERNS & PROCEDURES - I

Before you live in a house, it must be built. Before you can wear a dress, you must make it or buy it. Before you can drive a car, you must learn how to drive a car. Before you have a computer solve your problem, you must write the instructions for that problem.

PLANNING YOUR PROGRAM

In order to instruct the computer to do some task for you, you must provide the computer with a set of instructions telling it exactly what to do in a language both you and the computer can understand. That set of instructions is called a program. You may use a program someone else has written or you may use a program you have written. Writing computer programs is an important part in understanding computers.

Complex programs are often written by a team of people. No single person, for example, writes the large programs used in the NASA space projects, accounting programs for large corporations, or educational programs for teaching elementary students. That team effort must be directed by a plan, much like the construction of a house is directed by a plan of action.

Students would not likely write programs of high enough complexity to require a team effort; however, the planning of even simple programs helps to create logical programs and to reduce errors. Students in beginning computer classes, like students in beginning clothing, auto repairs, or metal shop classes, are expected to have a plan for the products they produce in class so that their activities will be organized and productive.

An engineer uses a plan called a blueprint. An accountant uses standard accounting rules. A writer often outlines his work. A computer programmer writes an algorithm. An algorithm is a set of instructions for doing something. Computer programmers use sets of instructions too, and they have special methods and plans for these, like blueprints, accounting rules, and outlines.

Once the computer programmer makes an algorithm, the programmer checks it very carefully to see if it makes sense and seems to do what he or she wants it to do. Frequently, one programmer will ask another programmer to check the algorithm. This is called desk-checking. When someone else checks your logic and algorithm, it is called a walkthrough. If it is thought to be okay, the programmer converts his program to a computer language. Most of the time and effort required to correctly write a program goes into the writing of the algorithm. The converting of the algorithm to a program is a relatively easy process.

USES OF DOCUMENTATION

A computer program, like many other inventions of man, is frequently found to need change to meet new conditions and other unforeseen

PAGE 38

circumstances. For this reason, programmers keep copies of their algorithms and provide descriptions of programs within the programs themselves. Also, complete instructions concerning how to use the program are provided by the programmer. The instructions, algorithms, and descriptions of the program are called documentation. Without good documentation, programs are often not useable and frequently not easily improved and, hence, become obsolete quickly. Often, better results are obtained when one can build on a program that already exists rather than starting at the beginning.

TYPES OF ALGORITHMS

Algorithms, as used by people writing computer programs, generally are done in one of two ways. The instructions may be listed in words which looks like an outline. This form of planning and documentation is called an outline or pseudocode. The logic and procedures may also be shown in pictorial form. These are called flowcharts.

SEQUENCE OF PATTERNS

The easiest of all programs to write are the programs which require only a single sequence of instructions. Look at the example below.

Outline Version

Flowchart Version

1. Store a number in a place called X

2. Store a number in a place called Y

3. Add X and Y and store the result in Z

4. Print the value of Z

5. END of Program
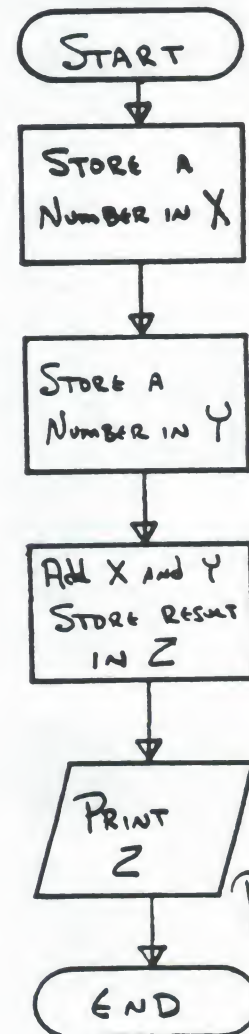
START

STORE A NUMBER IN X

STORE A NUMBER IN Y

ADD X AND Y STORE RESULT IN Z

PRINT Z

PAGE 39

END

BASIC Version

```
10   REM
20   REM   -- SAMPLE PROGRAM 1
30   REM
100  LET X = 4
110  LET Y = 12
120  LET Z = X + Y
130  PRINT "THE SUM IS ",Z
999  END
```

Definition of Variables

X - First number to be added

Y - Second number to be added

Z - Sum of two numbers

You can see that the outline version, the flowchart version, and the BASIC version convey similar information. The flowchart or pictorial representation of an algorithm is widely used by those who program computers. However, the outline version of writing algorithms is replacing the older flowchart method.

FLOWCHART SYMBOLS

There will be additional flowchart symbols, but the first four which we need to consider are these:

START and END symbol - There will only be one START. There will usually be only one END but there may be more than one. Use the same symbol with either START or END inside.

Processing symbol - This symbol shows when numbers need to be added or multiplied or when movement of data occurs. If you were to do these procedures, it would often require pencil and paper. When the computer does these procedures, it often requires the use of the arithmetic logic unit of the central processing unit.

Input and Output Symbol - Whenever the computer communicates with the outside world, this is the symbol you will use. When you need to have the computer give you an answer, it becomes an Output symbol. You will use words like Print, Write, and Put. When you need to have the computer get data, it becomes an Input symbol. Typical words used will be Get, Read, and Input.

Connector Symbol - This symbol exists only because you sometimes run out of paper when you

PAGE 40

draw flowcharts. This allows you to show where
the next step continues. It never appears on
an outline or in a BASIC program.

SEQUENCE PATTERNS

Suppose we limit our programming skills to these four flowcharting symbols.
The only programs we can write will be programs which are simple sequence
patterns. These are important problems, but they do not require the real
power of the computer. We will add the decision and looping patterns later.
Let's look at another example of a sequence pattern.

Outline Version

1. Read in a number and store it in
    a place called HOURS

2. Read in a number and store it in
    a place called RA

3. Multiply HOURS by RA and store it
    in a place called PAY

4. Print the HOURS

5. Print the RA

6. Print the PAY

7. END the Program

Flowchart Version

BASIC Version

JLIST

```
10   REM
20   REM   -- SAMPLE 2
30   REM
100   INPUT HOURS
110   INPUT RA
120   LET PAY = HOURS * RA
130   PRINT
140   PRINT "NUMBER OF HOURS WORKED THIS WEEK IS ";HOURS
150   PRINT "THE HOURLY RATE IS ";RA
160   PRINT "YOUR WEEKLY PAY IS $";PAY
999   END

JRUN
?31
?4.55

NUMBER OF HOURS WORKED THIS WEEK IS 31
THE HOURLY RATE IS 4.55
YOUR WEEKLY PAY IS $141.05
```
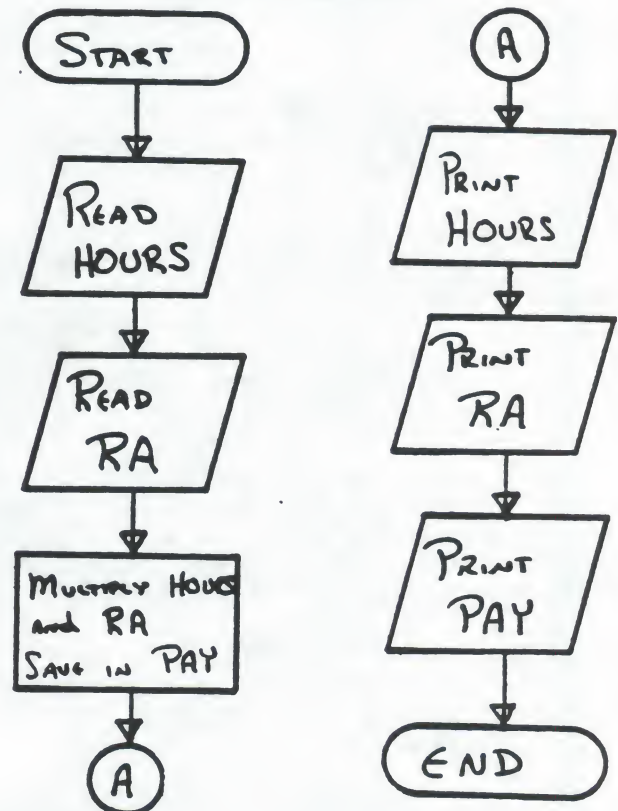
Definition of Variables

HOURS - Number of hours worked
per week

RA - Hourly rate   Page 41

PAY - Weekly paycheck amount

Some items need to be pointed out in SAMPLE 2.

1. RA was chosen for Rate because the reserved AT is in RATE. It is not possible to use a reserved word as part of a variable name.

2. Line 130 was not flowcharted or outlined. Some people might have included it in both places. In this case, the person who did the flowchart decided that line 130 had little to do with the logic of the program. It merely spaces the output in a better form.

3. The connector symbols are used in the flowchart. Any letter or number can be used, as long as it is clear.

4. The REM statements are never flowcharted or outlined.

5. The definition of variables includes a statement in words, describing the kind of data that is stored in each variable used in the program.

6. The procedure is described. No numbers occur in the logic of the program. This means that you can use different numbers every time you use the program. The program will do the same steps but with the numbers inserted at that time.

7. The START symbol on the flowchart never appears on an outline or in a program.

In the two examples which you have seen so far, both represent simple sequence patterns. Sequence patterns are a group of one or more program statements which are always done in the same order. When SAMPLE 1 program is done, the computer always does lines 10, 20, 30, 100, 110, 130, and 999 in that order. The Sequence is seven steps.

In a Sequence pattern, the computer automatically starts at the beginning and does one step at a time until it reaches the last statement. The REM statements are looked at, but nothing is done. The END statement stops the program.

The order of the steps are very important, because the computer does them in order. For example, in SAMPLE 1, lines 100 and 110 could be interchanged. However, line 120 could not be placed anywhere else without changing the results of the program.

SAMPLE 1 is an example of a program which looks like

PROCESSING ⟶ OUTPUT ⟶ END

SAMPLE 2 is an example of a program sequence which looks like

INPUT ⟶ PROCESSING ⟶ OUTPUT ⟶ END

PAGE 42

More than one step can be included in Input, Processing, and Output and they can occur in any order. The third example does not include any processing statements.

Outline Version

1. Print "Enter School Name"

2. Get SCHOOLS$

3. Print "Enter your name"

4. Get NAME$

5. Print NAME$, SCHOOL$

6. END the Program

Flowchart Version
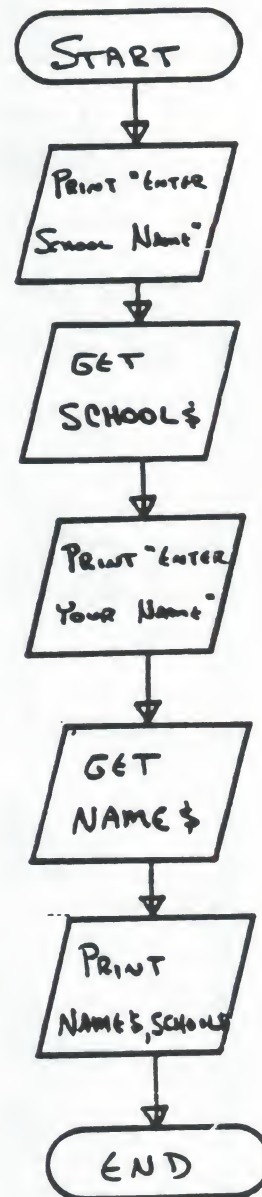
Definition of Variables

SCHOOL$ - Name of School

NAME$ - First name of student

BASIC Version

```
10   REM
20   REM  -- SAMPLE 3
30   REM
100  PRINT "ENTER YOUR SCHOOL NAME"
110  INPUT SCHOOL$
120  PRINT
130  PRINT "ENTER YOUR FIRST NAME"
140  INPUT FIRST$
150  PRINT
160  PRINT "HELLO, ";FIRST$;", FROM ";SCHOOL$
999  END

JRUN
ENTER YOUR SCHOOL NAME
?NALLWOOD JUNIOR HIGH

ENTER YOUR FIRST NAME
?DAVE

HELLO, DAVE, FROM NALLWOOD JUNIOR HIGH

JRUN
ENTER YOUR SCHOOL NAME
?BROADMOOR JUNIOR HIGH

ENTER YOUR FIRST NAME
?MELISSA

HELLO, MELISSA, FROM BROADMOOR JUNIOR HIGH
```


Flowchart: START → PRINT "Enter School Name" → GET SCHOOL$ → PRINT "Enter Your Name" → GET NAME$ → PRINT NAME$, SCHOOL$ → END

PAGE 43

Computer Literacy

VALUE OF CHANGE

Name: _____ Hour: _____ Date: _____

STATEMENT OF PROBLEM: Write a program that will give the dollar amount of 4
quarters, 5 dimes, 3 nickels, and 8 pennies. Assign the number of each type
of coin to a variable so that it would be easy to change the number of coins
without changing the formula or the PRINT statements.

BASIC STATEMENTS REQUIRED: REM, PRINT, LET, END

DOCUMENTATION REQUIRED: ◯ REM STATEMENTS   ◯ DEFINITION OF VARIABLES

Walkthrough by _____   ◯ FLOWCHART   ◯ OUTLINE OF SOLUTION

Code check by _____   ◯ PROGRAM LISTING   ◯ PROGRAM RESULTS

EXACT COPY OF RESULTS:

```
THE DOLLAR AMOUNT OF
4 QUARTERS
5 DIMES
3 NICKELS
8 PENNIES
IS $1.73
```

VALUE OF CHANGE

NOTE: The instructor will probably want the student to write a longer
flowchart. However, this one is a good example to discuss in class to
emphasize the desired sequence of steps.
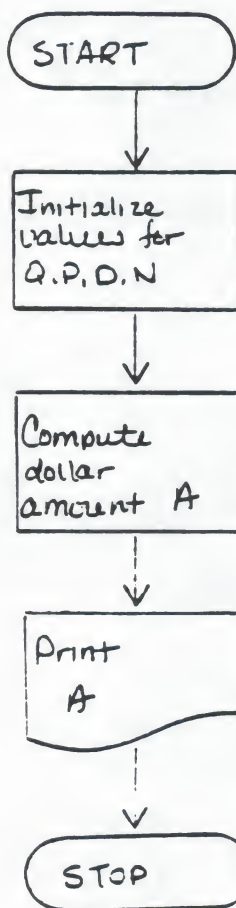
```
]LIST
10   REM   -- VALUE OF CHANGE
15   REM   -- WRITTEN BY KRISTI MEHRER
20   REM   -- OCTOBER, 1982
25   REM
41   REM   -- THIS PROGRAM WILL COMPUTE
42   REM   -- THE DOLLAR AMOUNT OF A
43   REM   -- NUMBER OF QUARTERS, DIMES,
44   REM   -- NICKELS, AND PENNIES
45   REM
100  LET Q = 4
110  LET D = 5
120  LET N = 3
130  LET P = 8
140  LET A = .25 * Q + .1 * D + .05 * N + .01 * P
150  PRINT "THE DOLLAR AMOUNT OF"
160  PRINT Q;" QUARTERS"
170  PRINT D;" DIMES"
180  PRINT N;" NICKELS"
190  PRINT P;" PENNIES"
200  PRINT "IS $";A
999  END

]RUN
THE DOLLAR AMOUNT OF
4 QUARTERS
5 DIMES
3 NICKELS
8 PENNIES
IS $1.73
```

Flowchart:
- START
- Initialize values for Q,P,D,N
- Compute dollar amount A
- Print A
- STOP

INPUT AND OUTPUT

In order for a computer to be good for anybody, it must communicate with the outside world. It communicates with the outside world in two ways. The first is by getting data. This is called input. The second is by sending information. This is called output. Different computers and different computer languages get and send information in a variety of ways. These are referred together as I/O. I/O stands for Input/Output operations. Input and output operations are very slow compared to the processing which takes place within the computer. We, as humans, think they are very fast, but I/O operations are really very time-consuming operations when they are compared to the internal speeds of a computer.

Using the Applesoft BASIC language, we will consider two ways to get information into a program and one way to print the results.

PRINT STATEMENTS

The PRINT statement is the output statement for BASIC. You have used this several times in some of the earlier study guides. The different uses of the PRINT statement will be reviewed here. Look at PRINT PROGRAM 1 below.

```
]LIST

10   REM  -- PRINT PROGRAM 1
20   REM
100   PRINT "THIS IS A SAMPLE OF A PRINT STATEMENT"
110   PRINT
120   PRINT "COLUMN 1","COLUMN 2","SUM"
130   LET X = 5
140   LET Y = 6
150   LET Z = X + Y
160   PRINT X,Y,Z
170   PRINT X;Y;Z
180   PRINT X;Y,Z
190   PRINT "THE SUM OF ";X;" AND ";Y;" IS ";Z
999   END

]RUN
THIS IS A SAMPLE OF A PRINT STATEMENT

COLUMN 1           COLUMN 2           SUM
5                  6                  11
5611
56                 11
THE SUM OF 5 AND 6 IS 11
```

Line 100 puts out the information in quotation marks, starting at the left hand side of the monitor or printer. Line 110 prints a blank line. This

PAGE 46

causes the printer or monitor to move to the next line before it prints out the next line of output. A programmer can use several of these PRINTs to leave spacing in the results.

Line 120 prints out several items with spacing between columns. When commas are used to separate output items, each item is allowed sixteen print positions. If most or all of the print positions are used, an item may use two or more columns of space to print the information.

Line 160 prints out the numeric answers in the same columns in which line 120 printed the headings.

Lines 170 and 180 use commas and semicolons. You will notice that semicolons place no spaces between the results, while commas automatically space in columns. You must be careful when you mix commas and semicolons.

Line 190 uses semicolons to give a printed result which is very readable and understandable. Notice the extra spaces inside the quotation marks to cause the exact spacing desired.

INPUT Statements

The INPUT statement is a BASIC statement which allows the computer user to enter data into a program while the program is being run. It occurs in two forms:

1. The word INPUT followed by one or more variable names separated by commas. Semicolons are not permitted.

2. The word INPUT followed by a string message of some kind in quotation marks. A semicolon follows the message and one or more variables are required. When more than one variable is written, commas must separate the variable names.

SAMPLE PROGRAM 1 below illustrates the first case and SAMPLE PROGRAM 2 illustrates the second case.

```
]LIST

10   REM   -- SAMPLE PROGRAM 1
20   REM
100   PRINT "ENTER A NUMBER"
110   INPUT A
120   PRINT "THE NUMBER YOU ENTERED WAS ";A
999   END

]RUN
ENTER A NUMBER
?50
THE NUMBER YOU ENTERED WAS 50
```

```
JLIST

10   REM   -- SAMPLE PROGRAM 2
20   REM
100   INPUT "ENTER A NUMBER ";A
120   PRINT "THE NUMBER YOU ENTERED WAS ";A
999   END

JRUN
ENTER A NUMBER 50
THE NUMBER YOU ENTERED WAS 50
```

SAMPLE PROGRAM 3 below illustrates all of the possible ways that an INPUT statement can be used.

```
JLIST

10   REM   -- SAMPLE PROGRAM 3
20   REM.
100   INPUT "ENTER TWO NUMBERS ";A,B
110   PRINT "NOW ENTER THREE MORE NUMBERS"
120   INPUT C,D,E
130   PRINT "ENTER ANOTHER NUMBER ";
140   INPUT F
150   PRINT "ENTER THE LAST NUMBER ",
160   INPUT G
170   LET SUM = A + B + C + D + E + F + G
180   PRINT
190   PRINT "THE SUM IS ";SUM
999   END

JRUN
ENTER TWO NUMBERS 1,2
NOW ENTER THREE MORE NUMBERS
?3,4,5
ENTER ANOTHER NUMBER ?6
ENTER THE LAST NUMBER          ?7

THE SUM IS 28
```

Line 100 asks for two numbers and allows the user to type them in on the same line as the question.

Lines 110 and 120 are treated together. The message NOW ENTER THREE MORE

PAGE 48

NUMBERS is typed in on one line and the user types in the desired numbers on the following line.

Lines 130 and 140 are a group and lines 150 and 160 are a group. The difference between these are the comma and the semicolon. The only difference is in the spacing of the message and the response. The commas allow a larger blank region before the question mark is printed. Study SAMPLE PROGRAM 3 carefully until you understand the use of the INPUT and the need for having a PRINT go with each INPUT.

SAMPLE PROGRAM 4 gives exactly the same results, but the program is not nearly as useful because the program does not help the user to know what he or she is expected to type in. The use of the PRINT statements makes the INPUT statement more useful.

```
]LIST

10   REM   -- SAMPLE PROGRAM 4
20   REM
100   INPUT A,B
120   INPUT C,D,E
140   INPUT F
160   INPUT G
170  .LET SUM = A + B + C + D + E + F + G
180   PRINT
190   PRINT "THE SUM IS ";SUM
999   END

]RUN
?1,2
?3,4,5
?6
?7

THE SUM IS 28
```

A SPECIAL CAUTION MUST BE MENTIONED WHEN YOU USE THE INPUT STATEMENT ON A PRINTER. The printer only prints entire lines at one time. If the number you type in and the message occur on the same line, then the printer will not print the message or the question mark until you write the answer and push the RETURN key. Then the whole line - message, question mark, and value - will be printed at the same time.

READ, DATA AND RESTORE STATEMENTS

The READ and DATA statements always work as a team. DATA statements can be placed anywhere in the program but it is suggested that you use the line numbers between 51 and 99. They should occur before the END statement.

All of the values listed in a DATA statement are listed in order inside the computer. Whenever a READ statement asks to read a value, the next one

is taken off of the list. Look at READ SAMPLE 1 below.

```
]LIST

10   REM   -- READ SAMPLE 1
20   REM
60   DATA  50,60,70,80
100  READ A
110  PRINT A
120  READ A
130  PRINT A
140  READ A,B
150  PRINT A,B
999  END

]RUN
50
60
70                      80
```

The rules which you must remember are:

1. If you use a READ, there must be at least one
    DATA statement in the program.

2. You may have more values in DATA statements than
    you have READ variables.

3. You may not have more variables in READs than you
    have values in DATA statements.

4. DATA values are separated with commas.

5. You may read either numeric or string values, but
    the variable names must match the kind
    of data read in.

6. When you use the same variable more than once in a
    READ, only the latest value is stored.

   READ SAMPLE 2 (below) illustrates some of the rules above. There is not
enough data left in the program by the time the program gets to line 170. The
OUT OF DATA ERROR IN 170 is printed to complain at that point.

```
]LIST

10   REM   -- READ SAMPLE 2
20   REM
60   DATA   5,7,9,11,13,15
70   DATA   17,19,21,23
80   DATA   25
100  READ A,A,A,A,A,A
110  PRINT A
120  READ B,C,D
130  PRINT A,B,C
140  PRINT D
150  READ A,E
160  PRINT A,B,E
170  READ F
180  PRINT F
999  END

]RUN
15
15              17              19
21
23              17              25
```

The next example, READ SAMPLE 3, shows how to mix the numeric and string variables and values. They must be matched correctly. One new statement is shown in this program. This is the RESTORE statement. It tells the computer to start from the beginning of the DATA statements again, regardless if it has completed the entire list or not.

```
]LIST

10    REM   -- READ SAMPLE 3
20    REM
60    DATA  5,"MATHEMATICS",8
70    DATA  "ENGLISH","HISTORY",3
80    DATA  "TOM AND JERRY"
100   READ A,B$,C,D$,E$
110   PRINT A,B$
120   READ F,G$
130   PRINT C,D$,E$
140   PRINT F,G$
150   RESTORE
160   PRINT
170   READ Z,Y$,X,Q$
180   PRINT Z,Y$
190   PRINT X,Q$
999   END

]RUN
5             MATHEMATICS
8             ENGLISH          HISTORY
3             TOM AND JERRY

5             MATHEMATICS
8             ENGLISH
```

You may use as many DATA statements as you want in a program but they
will be treated as one continuous statement.

ASSIGNING VALUES TO LOCATIONS

There are three ways to assign numeric and string values to variable
locations. These include the use of the LET statement, the READ statement,
and the INPUT statement. Variables must appear in one of these three kinds of
statements before they are used in other parts of the program.

The LET statement allows the value to be put into a variable location
within the program. The READ statement assigns a value which is read from a
DATA statement.

The INPUT statement accepts a value from the operator of the program.
The program stops and waits for the operator to enter the data.

The READ choice is useful when the data will be the same again and again
as the program is run repeatedly. The INPUT is the best choice when the data
will change each time the program is run. Frequently, both are used within
the same program because some data stays the same and other data changes.

Computer Literacy

CHOICES

The computer starts to become powerful when it is allowed to do different things depending upon the data inside. Suppose that you are asked to read in two numbers and print out the larger number. If both numbers are the same, then simply write out the first number. Computers make these choices in many different kinds of programs.

In this study guide, you will learn about the IF/THEN and the GO TO commands. These two statements allow the computer to make those choices for you. Look at CHOICES SAMPLE 1 below. This solves the problem suggested above.

```
JLIST

10   REM  -- CHOICES SAMPLE 1
20   REM  --
40   REM  -- THIS PROGRAM PRINTS OUT
42   REM  -- THE LARGER OF TWO NUMBERS.
44   REM  -- WHEN BOTH NUMBERS ARE THE
46   REM  -- SAME, THE FIRST IS PRINTED.
48   REM  --
100   INPUT "ENTER TWO NUMBERS:   ";A,B
109  REM
110  REM  -- CHOICE
111  REM
120   IF A >  = B THEN 160
129  REM
130  REM  -- FALSE CASE -- PRINT B
131  REM
140  PRINT B;" IS THE LARGER NUMBER."
150  GOTO 180
159  REM
160  REM  -- TRUE CASE  -- PRINT A
161  REM
170  PRINT A;" IS THE LARGER NUMBER."
179  REM
180  REM  -- END OF CHOICE
181  REM
999  END

JRUN
ENTER TWO NUMBERS:   6,9
9 IS THE LARGER NUMBER.

JRUN
ENTER TWO NUMBERS:   5,5
5 IS THE LARGER NUMBER.

JRUN
ENTER TWO NUMBERS:   4,1
4 IS THE LARGER NUMBER.
```

PAGE 53

Line 100 asks for and receives the two numbers which the user will type in. Line 120 checks to see if A is greater than or equal to B. When this relationship is TRUE then the computer proceeds to line 160 to continue with the directions. If A is not greater than or equal to B, then the next statement after line 120 is executed. This means line 140 is executed. Line 150 says to the computer that the next instruction is at line 180. That means that the computer skips some of the instructions and goes immediately to line 180.

CHOICES SAMPLE 2 is exactly the same program as CHOICES SAMPLE 1. The only difference (besides line 10) is that all unnecessary REM statements are deleted. It is much more difficult to read. Use the extra REM statements in your program to make them more readable.

```
]LIST

10   REM  -- CHOICES SAMPLE 2
40   REM  -- THIS PROGRAM PRINTS OUT
42   REM  -- THE LARGER OF TWO NUMBERS.
44   REM  -- WHEN BOTH NUMBERS ARE THE
46   REM  -- SAME, THE FIRST IS PRINTED.
48   REM  --
100   INPUT "ENTER TWO NUMBERS:   ";A,B
110   REM  -- CHOICE
120   IF A > = B THEN 160
130   REM  -- FALSE CASE -- PRINT B
140   PRINT B;" IS THE LARGER NUMBER."
150   GOTO 180
160   REM  -- TRUE CASE  -- PRINT A
170   PRINT A;" IS THE LARGER NUMBER."
180   REM  -- END OF CHOICE
999   END
```

THE GOTO STATEMENT

The GOTO statement has the fewest rules of all the different BASIC language statements. The GOTO statement must have a line number and it must tell which line number to GO TO.

The computer always prints the GOTO as one word. If you write it as two

PAGE 54

words, it still will squeeze it together and write it as one word.

The GOTO allows you to skip statements. This is very important when you want the computer to make a choice among several things.

CHOICES

When you use the IF/THEN and GOTO statements to create a choice, it must be composed of four parts:

1. The CHOICE -- This is the IF/THEN statement and it is identified with a REM statement before the IF/THEN statement.

2. The FALSE OPTION -- The False option always occurs immediately after the IF/THEN statement. The False Option always starts with the REM statement identifying that option. The GOTO statement is usually the last statement of the False Option.

3. The TRUE OPTION -- The True Option begins on the REM statement, identifying the True Option. The line number is the same as the line number at the end of the IF/THEN statement.

4. The END OF CHOICE -- This is the REM statement where both of the options end before the computer program starts on its next step.

Look at CHOICES SAMPLE 3 below. This program determines whether the four tests which a student takes allows him to get an A or not. An A is a grade of at least 94%. The program has been run twice. The results are different because the IF/THEN statement follows the TRUE or FALSE choice, depending if the average is equal to or greater than 94. The four parts of the choice are:

1. The CHOICE -- REM - line 130
              IF/THEN - line 140

2. The FALSE OPTION -- REM - line 150
                  Results - line 160, 170, 180

3. The TRUE OPTION -- REM - line 190
                 Results - line 200, 210

4. The END OF CHOICE -- REM - line 220

A Sequence occurs in front of the choice (lines 100, 110, 120), after the choice (lines 230, 240, 250, 260, 270, 280), and in both parts of the choice (lines 160, 170, 180 and lines 200, 210).

```
JLIST

10   REM   -- CHOICES SAMPLE 3
20   REM   --
40   REM   -- THIS PROGRAM TELLS THE USER
42   REM   -- IF SHE HAS RECEIVED
44   REM   -- AND GRADE OF A OR NOT.
46   REM   -- THE TEST SCORES ARE STORED
48   REM   -- IN THE DATA STATEMENT ON LINE  60
60   DATA  85,84,98,100
100   READ T1,T2,T3,T4
110   LET SUM = T1 + T2 + T3 + T4
120   LET AVE = SUM / 4
129   REM
130   REM   -- CHECK FOR AN 'A' GRADE
131   REM
140   IF AVE >  = 94 THEN 190
149   REM
150   REM   -- FALSE CHOICE - GRADE IS LESS THAN AN 'A'
151   REM
160   PRINT "THE AVERAGE WAS ";AVE
170   PRINT "THIS WAS NOT AN A AND YOU SHOULD WORK HARDER."
179   REM
180   GOTO 220
181   REM
190   REM   -- TRUE CHOICE - GRADE IS AN 'A' GRADE
191   REM
200   PRINT "SUPER!!  YOU RECEIVED AN A GRADE"
210   PRINT "YOUR AVERAGE WAS ";AVE
219   REM
220   REM   -- END OF CHOICE
221   REM
230   PRINT
240   PRINT "YOUR TEST SCORES WERE:"
250   PRINT   TAB( 12);T1
260   PRINT   TAB( 12);T2
270   PRINT   TAB( 12);T3
280   PRINT   TAB( 12);T4
999   END

JRUN
THE AVERAGE WAS 91.75
THIS WAS NOT AN A AND YOU SHOULD WORK HARDER.

YOUR TEST SCORES WERE:
            85
            84
            98
            100

160 DATA 90,96,98,95

JRUN
SUPER!!  YOU RECEIVED AN A GRADE
YOUR AVERAGE WAS 94.75

YOUR TEST SCORES WERE:
            90
            96
            98
            95
```

USE OF TAB

Lines 250-280 use a new command in the PRINT statement. The TAB command within a PRINT statement allows the programmer to space the output. In this case, the TAB(12) tells the computer to space 12 spaces before it prints the results. Unfortunately, the results are not exactly the same on both the monitor and the printer.

A. On the monitor, the information is printed in the column indicated or the next available space. For example,

   PRINT TAB(5); A; TAB(20); B

prints A starting in column 5 and B starting in column 20. Semi-colons must be used.

B. With the printer, the information is spaced over the required number of spaces before the results are printed. For example,

   PRINT TAB(5); A; TAB(20): B

will print A starting in column 5 and B starting in column 26 (if A is a one digit number). Semi-colons must be used.

When you use TAB as the first instruction in a PRINT statement, it does the same thing on both the monitor and the printer.

RELATIONAL OPERATORS

There are six relational operators you can use in the IF/THEN statement. These are described below. Any of these can be used in an IF/THEN statement.

SYMBOL	MEANING
=	EQUALS
<	LESS THAN
>	GREATER THAN
<=	LESS THAN OR EQUAL TO
>=	GREATER THAN OR EQUAL TO
<>	IS NOT EQUAL TO

MORE ABOUT THE IF/THEN

The IF/THEN statement is very flexible. You will not use many of these other options, but you should know that they exist:

1. You may have only one choice. If so, then either the TRUE statement or FALSE option goes immediately to the END OF CHOICE statement. The four REM statements are still required.

2. The PRINT, READ, LET and other statements may be used immediately at the end of the IF/THEN statement. This is not desirable, although it is possible. This is not desirable because the TRUE and FALSE options are not clear.

3. The use of the word AND can be used. All of the parts must be TRUE for the entire statement to be TRUE. For example,

        IF A = B AND C = D AND E = F THEN 400

   In this case, transfer will be made to statement 400 only when A = B and C = D and E = F. Otherwise, it will go to the next statement, the FALSE option.

4. The use of the word OR can be used. At least one of the parts must be TRUE for the entire statement to be TRUE. For example,

        IF A = B OR C = D OR E = F THEN 300

   In this case, transfer will be made to line 300 if at least one of the three parts are TRUE. Otherwise, if all three parts are FALSE, then the next statement, the FALSE option, will be taken.

5. The use of the word NOT can be used. For example,

        IF NOT A = B THEN 700

   Here, if A = B is NOT TRUE then it will transfer to line 700. If A = B is TRUE then the next statement, the FALSE option will be taken.

        The use of the NOT option is usually confusing and should be avoided by the beginning programmer.

    The five items listed above will be of interest to the serious programmer, but should be ignored by everybody else.

REMINDER ABOUT THE LIST COMMAND

    As you have noticed, the programs are getting longer. This is because many REM statements are being inserted in the program. Hence, many of your programs cannot be displayed on the monitor all at once.

    It will be useful for you to remember that the LIST command occurs in these forms.

LIST   The LIST by itself lists the entire program.

If the program is too long for the entire program to appear on the monitor, the first part will scroll off the top of the screen.

LIST 100, 200   This lists the lines in the program from the first line listed to the last line listed. This allows you to display the lines you wish on the screen.

LIST 200   This lists only the line requested on the screen.

PATTERNS & PROCEDURES - II

DECISION STRUCTURES

You have probably already figured out, if you didn't know, that the statements in a listed version of an algorithm are performed in the order listed, beginning at the top of the list. In the flowchart form of an algorithm, the statements are performed in the order indicated by following the arrows (or flow lines, as they are sometimes called) beginning at START. There are frequently occasions in which the standard order of starting at the beginning of a listed algorithm and proceeding in order to the end must be changed in order to accomplish the task required.

Two basic categories of changing the order of execution of statements in an algorithm involve choices and loops. Choice simply means that some statements of an algorithm may be done only when some conditions are true, and therefore they might not be done every time one performs the algorithm. The other category, loops, means that some statements in the algorithm would be done many times but probably would not be listed many times. The two categories, choices and loops, together are called decision structures. In the case of choices, decisions about which statements will be done must be made. In the case of loops, decisions about how long a set of statements will be done before the loop stops must be made.

We have already discussed Sequences. You will study Choices in ` study guide and loops will be discussed in Patterns & Procedures - III. are the only three kinds of structures required for programmers. Additional kinds of computer language statements just make these three easier to do.

FIRST CHOICE

There are three ways that Choices are commonly used. The first one is the one most frequently used. This is the IF...THEN...ELSE...END method. Look at the example below. Something is always done, but different things are done depending whether the IF/THEN statement is TRUE or FALSE.

```
JLIST

10   REM   -- PATTERNS SAMPLE 1
20   REM   --
22   REM   -- IF...THEN...ELSE...END METHOD
24   REM   --
40   REM   -- THIS PROGRAM DETERMINES IF AN  EMPLOYEE
42   REM   -- HAS WORKED OVERTIME HOURS OR NOT.
44   REM   --
100   INPUT "ENTER THE NUMBER OF HOURS WORKED:   ";HOURS
109   REM
110   REM   -- CHOICE - DETERMINE IF OVERTIME
111   REM
120   IF HOURS > 40 THEN 160
129   REM
130   REM   -- FALSE CHOICE - REGULAR PAY
131   REM
140   PRINT "REGULAR PAY"
150   GOTO 180
159   REM
160   REM   -- TRUE CHOICE - OVERTIME PAY
161   REM
170   PRINT "OVERTIME PAY"
179   REM
180   REM   -- END OF CHOICE
181   REM
999   END

JRUN
ENTER THE NUMBER OF HOURS WORKED:   30
REGULAR PAY

JRUN
ENTER THE NUMBER OF HOURS WORKED:   40
REGULAR PAY

JRUN
ENTER THE NUMBER OF HOURS WORKED:   50
OVERTIME PAY
```

In PATTERNS SAMPLE 1, the number of hours worked is read in, then either "REGULAR PAY" or "OVERTIME PAY" is written out. Exactly one is written out. The one which is written out depends on the value of HOURS.

The flowchart and the Definition of Variables are below. There are two ways to write the Outline. Both are shown.

Definition of Variables

HOURS - Number of hours worked per week

Outline Version I

1. Ask for the number of
   hours worked

2. Get the number of HOURS
   worked

3. IF HOURS is greater than 40

      THEN (TRUE part)

         Print "OVERTIME PAY"

      ELSE (FALSE part)

         Print "REGULAR PAY"

4. END of Program

Flowchart

Outline Version II

1. Ask for the number of HOURS
   worked

2. Read the number of HOURS
   worked

3. Is HOURS greater than 40?

                (FALSE)   (TRUE)

   A. Print "REGULAR PAY"   A. Print "OVERTIME PAY"

4. END of Program
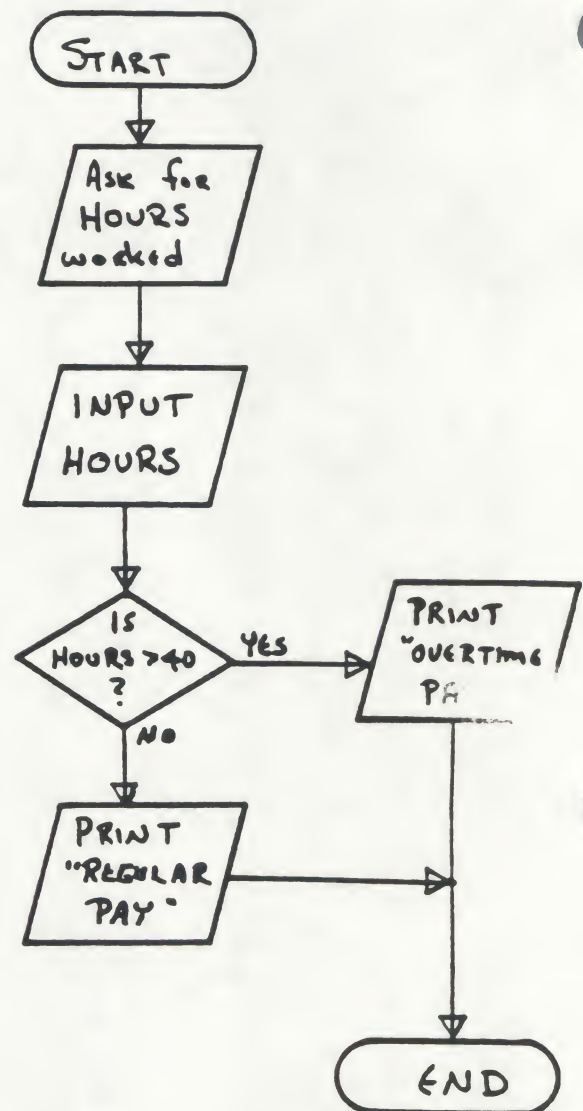
    In using the IF...THEN...ELSE...END version, the choice is determined in
the flowchart by a diamond shaped box. Something happens on each of the
pathways and they come together again after that something is done. The two
pathways can be labeled as either "TRUE" and "FALSE" or as "YES" and "NO."

    In Version I of the Outline, there are two choices -- The THEN part
(which is TRUE) and the ELSE part (which is FALSE). More than one instruction

PAGE 62

can be included in either or both parts. In Version II of the Outline, there are two columns of instructions. The computer will choose to do exactly one of those two columns. The columns would be labeled either "TRUE" and "FALSE" or "YES" and "NO."

SECOND CHOICE

The Second Choice is not used as frequently as the First Choice. It is called IF...THEN...END choice. There are still two pathways, but one goes directly to the END of the Choice because it doesn't require any action on one of the pathways. Look at the example below.

Suppose, as an example, that you enter a room with a large window and wish to remain in the room to read a book. If there is not enough light in the room coming through the window, you must turn on a lamp before you begin reading. A structured algorithm for this example would be the following:
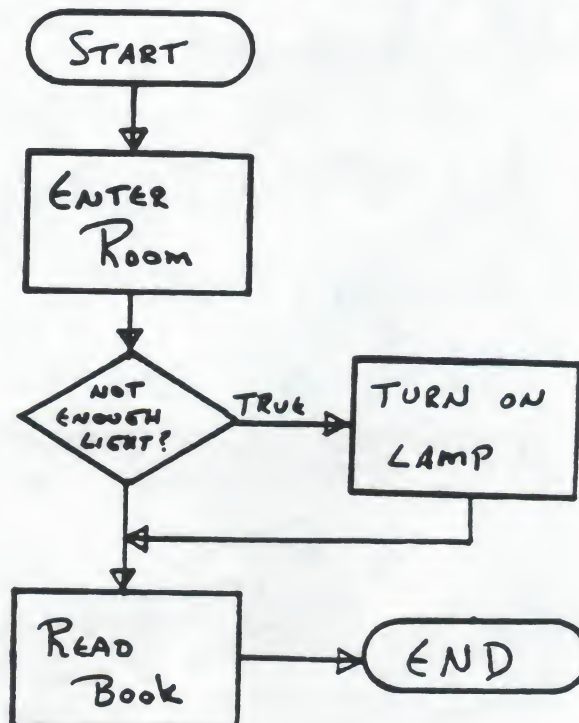
1. Enter the Room

2. IF there is not enough light in the room

        THEN

            Turn on the lamp

3. Begin reading the book

4. END of problem

The IF...THEN...END choice demands that you perform some statement only on some condition, but you do not do it otherwise. Here is the flowchart:

PAGE 63

This is not a computer example, but it illustrates that you must turn on the light before reading the book providing that the room does not have enough light and the lamp is not on.

Look at PATTERNS SAMPLE 2. Passing for this class is considered to be 70% or better. Failing is below 70%. The average is always computed and printed. The failing and progress report notice is printed only when a student has done failing work.

The program looks exactly like the IF...THEN...ELSE...END method, except that one of the choices does nothing.

The program has been run twice, once with a failing grade and once with a passing grade.

BASIC Version

```
LIST

10    REM   -- PATTERNS SAMPLE 2
20    REM   --
40    REM   -- THIS PROGRAM COMPUTES THE AVERAGE GRADE
42    REM   -- OF A STUDENT TAKING COMPUTER LITERACY.
44    REM   -- IT ALSO INDICATES WHEN A STUDENT IS FAILING
46    REM   -- AND A PROGRESS REPORT IS NECESSARY.
48    REM   --
60    DATA  50,70,90,45
100   READ S1,S2,S3,S4
110   LET AVE = (S1 + S2 + S3 + S4) / 4
119   REM
120   REM   -- CHOICE - DETERMINE IF FAILING GRADE
121   REM
130   IF AVE > = 70 THEN 180
139   REM
140   REM   -- FALSE CHOICE - FAILING GRADE
141   REM
150   PRINT "THIS STUDENT IS FAILING."
160   PRINT "SEND HOME A PROGRESS REPORT TODAY."
170   GOTO 190
179   REM
180   REM   -- TRUE CHOICE - NO ACTION REQUIRED
181   REM
189   REM
190   REM   -- END OF CHOICE
191   REM
200   PRINT "PRESENT AVERAGE IN COMPUTER LITERACY IS ";AVE
999   END

JRUN
THIS STUDENT IS FAILING.
SEND HOME A PROGRESS REPORT TODAY.
PRESENT AVERAGE IN COMPUTER LITERACY IS 63.75


160 DATA 80,90,60,70

JRUN
PRESENT AVERAGE IN COMPUTER LITERACY IS 75
```

Cl /Pattern - 2 Program Logic II/Page 5/11-82/s

Definition of Variables
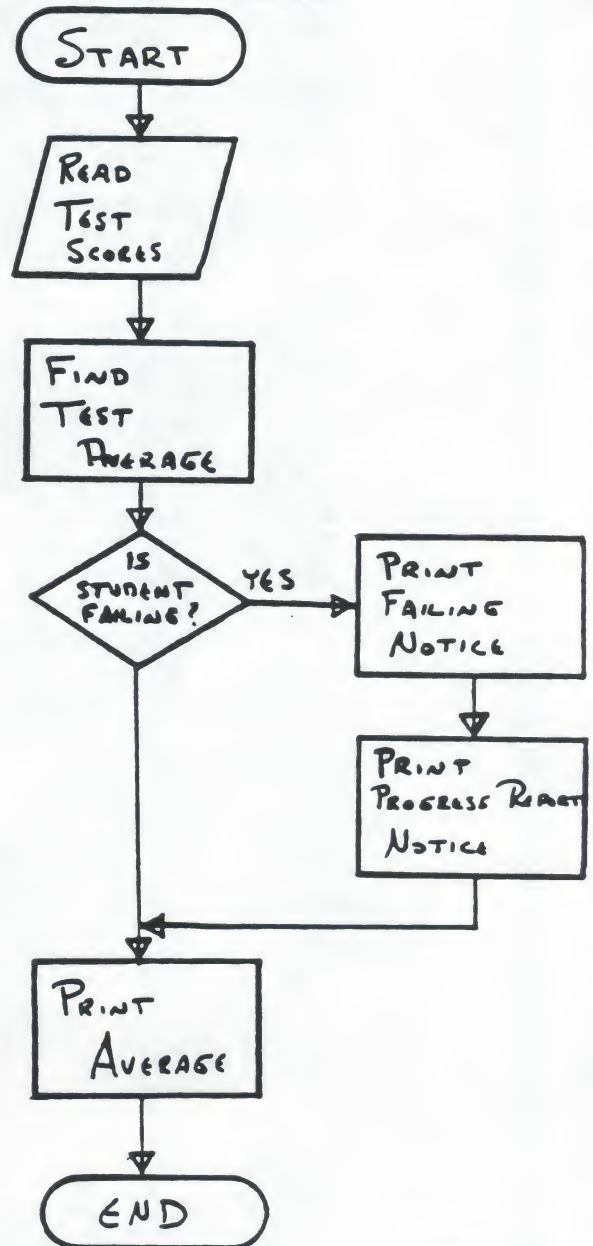
        S1 - Test Score #1
        S2 - Test Score #2
        S3 - Test Score #3
        S4 - Test Score #4
        AVE - Average of four tests

Outline Version I

1. Read Test Scores

2. Find Test Average

3. IF student is failing

    THEN

        Print Failing Notice
        Print Program Report Notice

4. Print Average

5. END the program

Flowchart Version

PAGE 65

Outline Version II

1. Read Test Scores

2. Find Test Average

3. Is student passing?

(NO)	(YES)
Print Failing Notice Print Program Report Notice	Do Nothing

4. Print Average

5. END the program

On the Flowchart Version, you will see that either something is done or nothing is done before the average is printed.

THIRD CHOICE

The third Choice is used when you want to pick one from many options. This example program will allow the user to type in the grade level that they are in school. If the user answers 1 through 6, the computer will answer "ELEMENTAY SCHOOL". If the user answers 7 or 8, the computer will answer "MIDDLE SCHOOL". If the user answers 9 through 12, the computer will answer "HIGH SCHOOL". If the student answers higher than 12, then "COLLEGE" will be printed. Any other response, like a negative number, will give an answer of "ERROR IN INPUT". Look at the next example, PATTERNS SAMPLE 3.

PAGE 66

```
]LIST

10   REM  -- PATTERNS SAMPLE 3
20   REM  --
40   REM  -- THIS PROGRAM ASKS FOR THE GRADE LEVEL
41   REM  -- AND THEN CLASSIFIES ACCORDING TO THE FOLLOWING:
42   REM  ------ BELOW 1    -- INPUT ERROR
43   REM  ------ 1 THRU 6   -- ELEMENTARY
44   REM  ------ 7 THRU 8   -- MIDDLE SCHOOL
45   REM  ------ 9 THRU 12  -- HIGH SCHOOL
46   REM  ------ OVER 12    -- COLLEGE STUDENT
48   REM  --
100   INPUT "ENTER YOU GRADE:   ";GADE
109   REM
110   REM  -- CHOICE - DETERMINE GRADE LEVEL
111   REM
120   IF GADE > 12 THEN 280
130   IF GADE >  = 9 AND GADE <  = 12 THEN 250
140   IF GADE = 7 OR GADE = 8 THEN 220
150   IF GADE >  = 1 AND GADE <  = 6 THEN 190
159   REM
160   REM  -- ERROR CHOICE
170   PRINT "ERROR IN INPUT VALUE."
180   GOTO 300
189   REM
190   REM  -- ELEMENTARY CHOICE
200   PRINT "YOU ARE AN ELEMENTARY STUDENT."
210   GOTO 300
219   REM
220   REM  -- MIDDLE SCHOOL CHOICE
230   PRINT "YOU ARE A MIDDLE SCHOOL STUDENT."
240   GOTO 300
249   REM
250   REM  -- HIGH SCHOOL CHOICE
260   PRINT "YOU ARE A HIGH SCHOOL STUDENT."
270   GOTO 300
279   REM
280   REM  -- COLLEGE CHOICE
290   PRINT "YOU ARE A COLLEGE STUDENT."
299   REM
300   REM  -- END OF CHOICES
999   END

]RUN
ENTER YOU GRADE:   4
YOU ARE AN ELEMENTARY STUDENT.

]RUN
ENTER YOU GRADE:   9
YOU ARE A HIGH SCHOOL STUDENT.

]RUN
ENTER YOU GRADE:   0
ERROR IN INPUT VALUE.
```

PAGE 67

Definition of Variables

GADE - Grade level of student
       (GRADE would have been preferred, but GR is a
       reserved word.)

Outline Version

1. Ask for the Grade level

2. Input the Grade level

3. IF Grade is greater than 12

        THEN

                Print "COLLEGE"
                GOTO Step 4

   IF Grade is between 9 and 12

        THEN

                Print "HIGH SCHOOL"
                GOTO Step 4

   IF Grade is between 7 or 8

        THEN

                Print "MIDDLE SCHOOL"
                GOTO Step 4

   IF Grade is between 1 and 6

        THEN

                Print "ELEMENTARY SCHOOL"
                GOTO Step 4

        ELSE

                Print "PRINT ERROR IN INPUT"

4. END of Program

PAGE 68
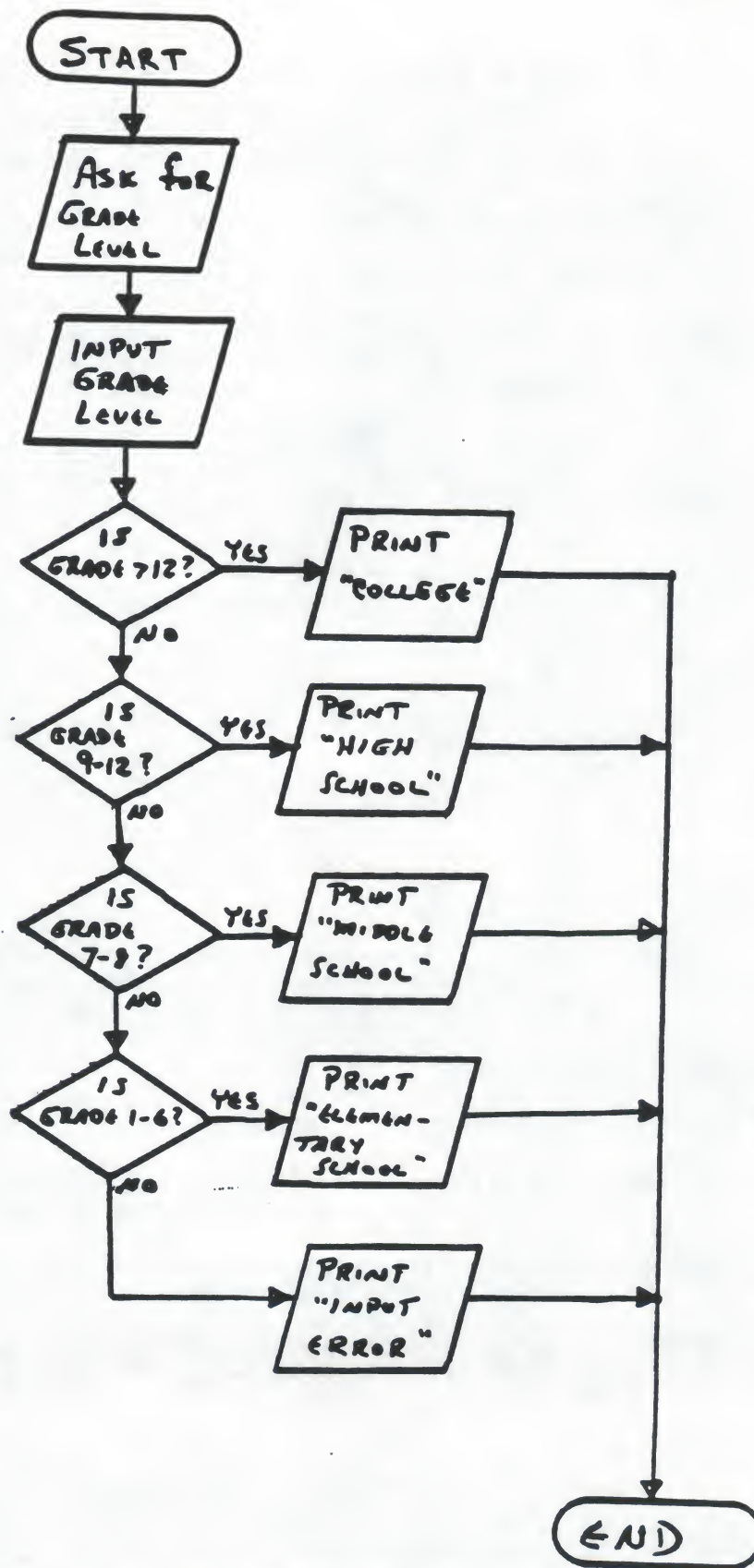
These rules apply to the Third Choice. This Third Choice is called the CASES Choice.

1. All of the IFs occur at the beginning of the Choice.

2. All choices end at the same point.

3. REMs are used to identify each separate case.

The use of the variable GRADE would have been preferred but GR is a reserved word and so it could not be used. Some of the REMs have been omitted so that the whole program would fit on one page.

The Choice question (lines 120 through 150) could have been written like the Sample below. You should understand that this serves the same purpose when whole numbers are typed in response to the question.

```
109   REM
110   REM  -- CHOICE - DETERMINE GRADE LEVEL
111   REM
120   IF GADE > 12 THEN 280
130   IF GADE > 8 THEN 250
140   IF GADE > 6 THEN 220
150   IF GADE > 0 THEN 190
```

SUMMARY OF FLOWCHARTING OF CHOICES

As a review, you will notice that a choice always uses the diamond-shape box in flowcharts. The paths out always are labeled "TRUE" and "FALSE" or "YES" and "NO". GOTO statements never are flowcharted.

REVIEW OF CHOICE STATEMENTS

Choice statements always have four parts:

1. The CHOICE - There may be more than one IF statement.
   However, all the IF statements should appear together.

2. The FALSE CHOICE - This comes immediately after the IF
   statement. There is always one false branch.

3. The TRUE CHOICE - This is the statement sent to when the IF
   statements are true. There may be more than one
   true choice.

4. The END OF CHOICE - This is where all of the choices come
   together at the end.

All of the parts of a choice must be clearly indicated with REM statements.

Computer Literacy

Name:_____ Hour:_____ Date:_____

STATEMENT OF PROBLEM: Write a program that will read in the numerators and
denominators of 2 fractions and will determine if the two fractions are >, <
or =. The items in the DATA statement should read in order first numerator,
first denominator, second numerator, second denominator.

BASIC STATEMENTS REQUIRED: REM, READ, DATA, IF/THEN, PRINT, END

SPECIAL PROBLEMS: Run the program three times so that each of the kinds of
output will be checked.

DOCUMENTATION REQUIRED: ◯ REM STATEMENTS   ◯ DEFINITION OF VARIABLES

Walkthrough by _____   ◯ FLOWCHART   ◯ OUTLINE OF SOLUTION

Code check by _____   ◯ PROGRAM LISTING   ◯ PROGRAM RESULTS

EXACT COPY OF RESULTS:

```
]80 DATA 3,4,6,8

]RUN
3/4=6/8

]80 DATA 1,3,2,3

]RUN
1/3<2/3

]80 DATA 5,7,3,7

]RUN
5/7>3/7
```

PAGE 71

COMPARING FRACTIONS

```
]LIST

10   REM   -- COMPARING FRACTIONS
15   REM   -- WRITTEN BY KRISTI MEHRER
20   REM   -- DECEMBER, 1982
25   REM
40   REM   -- THIS PROGRAM WILL READ IN THE
42   REM   -- NUMERATORS AND DENOMINATORS OF
44   REM   -- TWO FRACTIONS AND THEN DETERMINE
46   REM   -- IF THEY ARE =, < OR >
80   DATA  3,4,6,8
100  READ N1,D1,N2,D2
109  REM
110  REM   -- CHOICE - DETERMINE IF =, < OR >
111  REM
120  IF N1 / D1 > N2 / D2 THEN 200
130  IF N1 / D1 < N2 / D2 THEN 170
139  REM
140  REM   -- "=" OPTION
141  REM
150  PRINT N1;"/";D1;"=";N2;"/";D2
160  GOTO 220
169  REM
170  REM   -- "<" OPTION
171  REM
180  PRINT N1;"/";D1;"<";N2;"/";D2
190  GOTO 220
199  REM
200  REM   -- ">" OPTION
201  REM
210  PRINT N1;"/";D1;">";N2;"/";D2
219  REM
220  REM   -- END OF CHOICE
221  REM
999  END


]80 DATA 3,4,6,8

]RUN
3/4=6/8

]80 DATA 1,3,2,3

]RUN
1/3<2/3

]80 DATA 5,7,3,7

]RUN
5/7>3/7
```
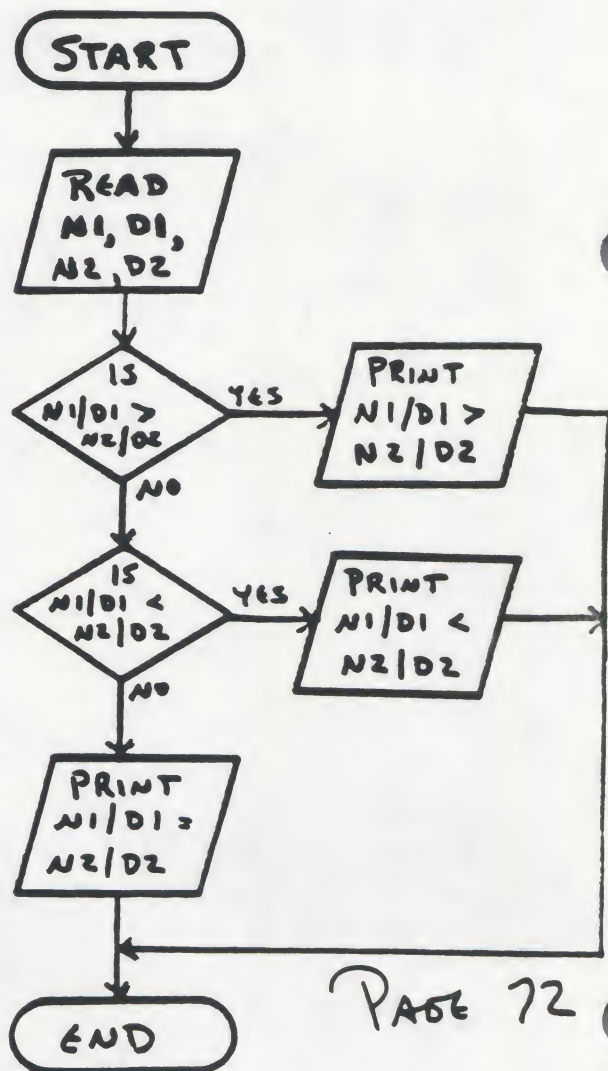
CL/12-82/a

PAGE 72

LOOPS

If Choices allow the computer to be powerful, then Loops permit it to be very, very powerful. Looping causes the computer to repeat some steps of a program again and again. Sometimes the same thing is done, and sometimes the steps which are executed change each time it is done. Let's look at LOOPS SAMPLE 1 below. It prints the same thing forever. It is called an infinite loop. This loop does not have an end to it. To stop the program, you must use the CTRL-C command.

```
JLIST

10   REM  -- LOOPS SAMPLE 1
20   REM  --
40   REM  -- THIS PROGRAM USES A LOOP
42   REM  -- TO PRINT OUT THE WORD
44   REM  -- "HELLO  " AGAIN AND AGAIN.
46   REM  -- THIS IS AN INFINITE LOOP.
48   REM  --
109  REM
110  REM  -- BEGINNING OF LOOP
111  REM
120  PRINT "HELLO  ";
130  GOTO 120
999  END


JRUN
HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO
 HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO
  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELL
O  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HEL
LO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HELLO  HE
LLO
BREAK IN 120
```

Most programs should not have infinite loops. This means we must know how to stop a loop. The IF/THEN statement is one way to stop a loop.

THE PARTS OF A LOOP

There are generally six parts to a loop. Sometimes, some steps are included with others so that there are really fewer steps. These steps are:

1. INITIALIZATION OF LOOP -- These are steps which are required for the loop to work correctly, but which need to be done before the loop. They are done only once.

2. START OF LOOP -- This is where the loop actually starts and continues to repeat the second and third times.

PAGE 73

3. INCREMENT OF LOOP -- This step keeps track of which time you are in the loop.

4. THE CHECK FOR END OF LOOP -- Determination is made to see if you need to repeat the loop or stop it.

5. REPEAT LOOP -- If the loop is to be repeated, this step sends the computer back to the START OF LOOP.

6. END OF LOOP -- This is where the computer goes when the loop is finished.

LOOPS SAMPLE 2 prints out the word HELLO exactly ten times. Each of the six parts are illustrated. All steps in a loop should be in the order listed, except for steps 3 and 4. Depending upon the program, sometimes these are in reverse order.

]LIST

```
10   REM  -- LOOPS SAMPLE 2
20   REM  --
40   REM  -- THIS PROGRAM USES A LOOP
42   REM  -- TO PRINT OUT THE WORD
44   REM  -- "HELLO " EXACTLY 10 TIMES.
46   REM  --
109  REM
110  REM  -- INITIALIZATION OF LOOP
111  REM
120  LET I = 0
129  REM
130  REM  -- START OF LOOP
131  REM
140  PRINT "HELLO  ";
149  REM
150  REM  -- INCREMENT OF LOOP
151  REM
160  LET I = I + 1
169  REM
170  REM  -- CHECK FOR END OF LOOP
171  REM
180  IF I = 10 THEN 210
189  REM
190  REM  -- REPEAT LOOP
191  REM
200  GOTO 130
209  REM
210  REM  -- END OF LOOP
211  REM
999  END
```

]RUN
HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO

PAGE 74

LOOPS SAMPLE 3 prints out the word HELLO eight times. You will notice that all six steps are shown, but that some BASIC statements do more than one step. This program uses the FOR/NEXT statement. It is a special pair of BASIC statements which make the creation of loops very easy.

You will probably always use the FOR/NEXT statements instead of the loop shown in LOOPS SAMPLE 2.

```
JLIST

10   REM   -- LOOPS SAMPLE 3
20   REM   --
40   REM   -- THIS PROGRAM USES A LOOP
42   REM   -- TO PRINT OUT THE WORD
44   REM   -- "HELLO " EXACTLY 8 TIMES
46   REM   -- USING A FOR/NEXT LOOP.
48   REM   --
109  REM
110  REM   -- INITIALIZATION OF LOOP
111  REM
120  FOR I = 1 TO 8
129  REM
130  REM   -- START OF LOOP
131  REM
140  PRINT "HELLO   ";
149  REM
150  REM   -- INCREMENT AND CHECK FOR END OF LOOP
151  REM   -- REPEAT LOOP, IF NOT DONE
152  REM
160  NEXT I
169  REM
170  REM   -- END OF LOOP
171  REM
999  END

JRUN
HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
```

THE FOR STATEMENT

The FOR statement always looks like

line #   FOR   variable   =   number1 TO number2 STEP number3

Number1 is always the value of the variable at the beginning of the loop. Number1 is often called the counter. Number2 is usually the ending value of the loop. Number3 is always the amount added, or subtracted, each time to the variable. The last two parts (STEP number3) is not required in the FOR statement. Some examples are below.

PAGE 75

```
100    FOR I = 1 TO 250
110    FOR J = 3 TO 85
120    FOR K = 27 TO 10 STEP - 1
130    FOR L = 1 TO 10 STEP .5
140    FOR M = 0 TO - 32 STEP - 2
150    FOR N = - 5 TO 5
160    FOR P = 1 TO 15000 STEP 8
```

When the last number (the increment) is not included, it will automatically be 1.

Line 100 will automatically start from 1, go to 250, and be incremented by 1 each time. The loop will occur 250 times.

Line 110 will go from 3 to 85, a total of 83 times through the loop.

Line 120 will start at 27 and go backwards until it gets to 10, a total of 18 times.

Line 130 will start at 1, increase by one-half each time, and end at 10. There will be a total of 21 times through the loop.

Line 140 will start at zero and go to -32, decreasing by -2 each time. There will be 17 times through the loop.

Line 150 will start at -5 and go to 5, increasing by 1 each time. There will be 11 times through the loop.

Line 160 will cause the loop to take place 1875 times, starting at 1 and going to 14993. Each time the loop is done, P will increase by 8.

```
]LIST

 10   REM   -- LOOPS SAMPLE 4
 20   REM   --
 40   REM   -- THIS PROGRAM USES A LOOP
 42   REM   -- TO PRINT OUT THE WORD
 44   REM   -- "INVENTORY ITEM # " FOLLOWED
 46   REM   -- BY THE NUMBER OF THE ITEM.
 48   REM   --
109   REM
110   REM   -- INITIALIZATION OF LOOP
111   REM
120   FOR I = 3 TO 25 STEP 4
129   REM
130   REM   -- START OF LOOP
131   REM
140   PRINT "INVENTORY ITEM # ";I
149   REM
150   REM   -- INCREMENT AND CHECK FOR END OF LOOP
151   REM   -- REPEAT LOOP, IF NOT DONE
152   REM
160   NEXT I
169   REM
170   REM   -- END OF LOOP
171   REM
999   END

]RUN
INVENTORY ITEM # 3
INVENTORY ITEM # 7
INVENTORY ITEM # 11
INVENTORY ITEM # 15
INVENTORY ITEM # 19
INVENTORY ITEM # 23
```

Look at LOOPS SAMPLE 4 above. Line 120 tells the computer to repeat the instructions through line 160. It will start with I equal to 3. 4 will be added to I each time. The loop will stop when I is equal to 25 or the smaller number closest to 25.

THE NEXT STATEMENT

The NEXT statement always looks like

line #   NEXT   variable

The NEXT statement must have the variable of the FOR statement with which it is matched. It tells the loop where to stop, where to increment before it starts again, and where to check to see if the loop is done. Some examples are:

PAGE 77

```
200    NEXT I
210    NEXT J
220    NEXT Z
```

Line 200 stops the loop with the variable I. Line 210 stops the loop with J and line 220 stops the loop with variable Z.

A FOR/NEXT EXAMPLE

Our problem is to read in 10 numbers and find the square of each of the numbers. The numbers are in a DATA statement. The program is below.

A heading is first put on top of each column. The loop causes each of the ten numbers to be read in from the DATA statement, find the square of the number, and write the number and the square out.

```
]LIST

10   REM  -- LOOPS SAMPLE 5
20   REM  --
40   REM  -- THIS PROGRAM USES A LOOP
42   REM  -- TO READ IN TEN NUMBERS AND
44   REM  -- PRINT THE SQUARE AND THE SQUARE
46   REM  -- OF THE NUMBER.
48   REM  --
80   DATA  45,23,890,3,275
85   DATA  -43,0,.41942,5000,-12
100   PRINT "NUMBER","SQUARE OF NUMBER"
105   PRINT
109   REM
110   REM  -- INITIALIZATION OF LOOP
111   REM        .
120   FOR A = 1 TO 10
129   REM
130   REM  -- START OF LOOP
131   REM
140   READ NUMBER
150   LET SQUARE = NUMBER * NUMBER
160   PRINT NUMBER,SQUARE
169   REM   ·
170   REM  -- INCREMENT AND CHECK FOR END OF LOOP
171   REM  -- REPEAT LOOP, IF NOT DONE
172   REM
180   NEXT A
189   REM
190   REM  -- END OF LOOP
191   REM
999   END

]RUN
NUMBER             SQUARE OF NUMBER

45                 2025
23                 529
890                792100
3                  9
275                75625
-43                1849
0                  0
.41942             .175913136
5000               25000000
-12                144
```

The variable A goes from 1 to 10 in this loop. It is used to make sure
that the loop is only done 10 times.

ANOTHER FOR/NEXT EXAMPLE

This time the problem is to add each of the numbers from 1 to 13 and write out each subtotal. The example below shows that the heading is written out (lines 100, 110), the SUM is set to zero (line 130), the loop repeats thirteen times (lines 140 through 190), and then stops.

Inside the loop, the next number is created (line 140), the number is added to SUM (line 160), the results are printed out (line 170), and the checks are made (line 190).

Every FOR statement must have a NEXT statement.

```
]LIST

10   REM  -- LOOPS SAMPLE 6
20   REM  --
40   REM  -- THIS PROGRAM USES A LOOP
42   REM  -- SUM THE NUMBERS FROM 1 TO 13
44   REM  -- WITH SUBTOTALS WRITTEN OUT.
46   REM  --
100   PRINT "NUMBER","SUBTOTAL SO FAR"
110   PRINT
119   REM
120   REM  -- INITIALIZATION OF LOOP
121   REM
130   LET SUM = 0
140   FOR B = 1 TO 13
149   REM
150   REM  -- START OF LOOP
151   REM
160   LET SUM = SUM + B
170   PRINT B,SUM
179   REM
180   REM  -- INCREMENT AND CHECK FOR END OF LOOP
181   REM  -- REPEAT LOOP, IF NOT DONE
182   REM
190   NEXT B
199   REM
200   REM  -- END OF LOOP
201   REM
999   END
```

```
]RUN
NUMBER                 SUBTOTAL SO FAR

1                      1
2                      3
3                      6
4                      10
5                      15
6                      21
7                      28
8                      36
9                      45
10                     55
11                     66
12                     78
13                     91
```

Computer Literacy

PATTERNS & PROCEDURES - III

We have looked at Sequences in Patterns & Procedures - I and Choices in Patterns & Procedures - II. This study guide will look at Loops. These three structures are the only three structures required to write programs in any language. Different computer languages use them differently. Some computer languages make them easy to use; others make them more difficult to use. We will illustrate how these are documented. In nearly all cases, you will find the FOR/NEXT version to be the easiest to use. However, the last example will show the IF/THEN loop.

PARTS OF A LOOP

All loops must consider six separate parts. Not all parts are required in all loops. Loops allow you to repeat certain parts of the programs. Stopping the loop at the right time is one of the major problems of looping. Frequently one step can handle several parts of the loop at once. The six parts of the loop are:

1. INITIALIZATION OF LOOP -- These are steps which are done <u>before</u> the loop is started. It needs to be done only once. Most loops require one or more statements to accomplish this. The FOR statement is one example.

2. START OF LOOP -- This is the beginning of the loop where the instructions start to repeat each time.

3. INCREMENT OF LOOP -- Occasionally this is not needed. In most loops, it is the statement which keeps track of how many times it has been through the loop. A FOR statement, for example, usually increments by one until the end is reached.

4. CHECK FOR END OF LOOP -- All loops must have this step unless it is an infinite loop. It can be an IF statement or a check to see if the increment variable is at the end.

5. REPEAT LOOP -- This step sends the computer back to the start of the loop to repeat the instructions again. The NEXT statement or a GOTO statement does this.

6. END OF LOOP -- When the loop is complete, the next program statement must be executed. This is usually the next statement after the parts of the loop. When the statement is reached, the loop is not repeated again.

FOR/NEXT VERSION OF LOOP

The FOR/NEXT version of a loop is the easiest to use. The first example will create and write out the numbers from one to five and the square roots of these numbers. The BASIC version, flowchart, outline version, and definition

PAGE 81

of variables follow.

Definition of Variables

Number - Loop counter number to find the square root of

Root - Square root of NUMBER

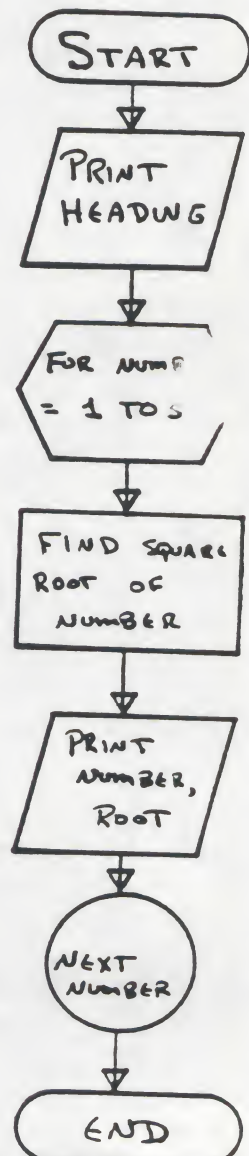## Functions Used

SQR - finds the square root of numbers

Outline

1. Print heading

2. LOOP 5 times:

   A. Find the square root of the number

   B. Print out the number and the square root

3. END of Program

BASIC Version

JLIST

```
10   REM  -- LOOPS - PROBLEM 1
20   REM  --
40   REM  -- THE PROBLEM CREATES THE NUMBERS
42   REM  -- FROM ONE TO FIVE AND PRINTS OUT
44   REM  -- THE NUMBERS AND THEIR SQUARE ROOTS.
46   REM
100   PRINT "NUMBER","SQUARE ROOT"
110   PRINT
119   REM
120   REM  -- INITIALIZATION OF LOOP
121   REM
130   FOR NUMBER = 1 TO 5
139   REM
140   REM  -- START OF LOOP
141   REM
150   LET ROOT =  SQR (NUMBER)
160   PRINT NUMBER,ROOT
169   REM
170   REM  -- INCREMENT AND CHECK FOR END OF LOOP
171   REM  -- REPEAT LOOP, IF NOT DONE
172   REM
180   NEXT NUMBER
189   REM
190   REM  -- END OF LOOP
191   REM
999   END
```

PAGE 82

```
]RUN
NUMBER          SQUARE ROOT

1               1
2               1.41421356
3               1.73205081
4               2
5               2.23606798
```

In PROBLEM 1, you will notice that the FOR statement through the NEXT statement is where the loop is. All of the steps in between are repeated five times, using a different number each time. Line 130 starts NUMBER at 1 and increases it by 1 until 5 is reached. When 5 is reached, the line following the NEXT statement (line 180) is executed.

The flowchart has a separate kind of box for the FOR statement and a large circle for the NEXT statement. These, and the statements between them, act as a group. You may never go from the outside of a loop into the middle of a loop without starting at the beginning.

The outline is written so that it is very clear what all of the steps of the loop are and what is repeated.

The "Functions Used" has been included in the Definition of Variables because SQR automatically uses some instructions inside the computer to find the square root of a number. Computers usually have many of these special instructions which allow you to use the computer easier and better.

ANOTHER FOR/NEXT EXAMPLE

As a second example, the problem is to find the sum of six numbers in a DATA statement, printing out each number as it is read in.

Definition of Variables

SUM - Sum of Numbers

Z - Loop counter

NUMBER - Value of each number to be added together

Outline

1. Print heading

2. Set SUM equal to zero

3. LOOP 6 times

PAGE 83

A. Read in next number

B. Print number out

C. Add number to SUM

4. Print out the SUM of the numbers

5. END the program

<u>BASIC Version</u>

```
]LIST

10    REM   -- LOOPS - PROBLEM 2
20    REM   --
40    REM   -- THE PROBLEM READS IN SIX NUMBERS,
42    REM   -- PRINTS THEM OUT, ADDS THE NUMBERS
44    REM   -- TO OBTAIN A SUM, AND THEN PRINTS
46    REM   -- OUT THE SUM AT THE END OF THE PROGRAM.
48    REM
85    DATA   45,129,930,3453,274,336
100   PRINT "THE NUMBERS TO BE ADDED ARE:"
109   REM
110   REM   -- INITIALIZATION OF LOOP
111   REM
120   LET SUM = 0
130   FOR Z = 1 TO 6
139   REM
140   REM   -- START OF LOOP
141   REM
150   READ NUMBER
160   PRINT  TAB( 8);NUMBER
170   LET SUM = SUM + NUMBER
179   REM
180   REM   -- INCREMENT AND CHECK FOR END OF LOOP
181   REM   -- REPEAT LOOP, IF NOT DONE
182   REM
190   NEXT Z
199   REM
200   REM   -- END OF LOOP
201   REM
210   PRINT
220   PRINT "THE SUM OF THE NUMBERS IS ";SUM
999   END

]RUN
THE NUMBERS TO BE ADDED ARE:
        45
        129
        930
        3453
        274
        336

THE SUM OF THE NUMBERS IS 5167
```
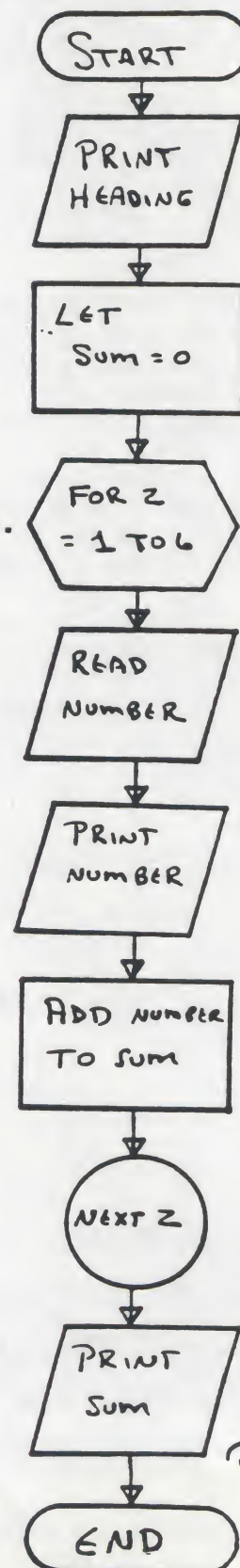
<u>Flowchart Version</u>

PAGE 84

In the flowchart of PROBLEM 2, you should notice that SUM must be initialized before the loop starts. This is because the NUMBER is added to the SUM inside the loop everytime NUMBER is read in.

The loop goes from line 130 through line 190 and the flowchart clearly shows the beginning and ending of the loop. In this program, the value for Z is not used except to keep track of how many times the loop is executed. When it has been done six times, the loop automatically stops. Each time the loop is executed the <u>next</u> number is pulled out of the DATA statement and assigned to NUMBER.

Look at the Outline carefully for this problem. It shows that the program goes through five major steps. Step 3 is the loop and the three parts of it are repeated until it is done.

IF/THEN VERSION OF LOOP

As a last example, the problem is not clearly stated at the beginning exactly how many times the loop is executed. The problem is to read in the numbers until a negative number is read in. Add all of the numbers, but not the negative number, and print out the average of the numbers.

Definition of Variables

SUM - The sum of the numbers read in
COUNT - The number of numbers read in
NUMBER - Each of the numbers read in
AVERAGE - The average of the numbers

Outline

1. Let SUM equal to zero

2. Let COUNT equal to zero

3. LOOP

    A. Read in the next number

    B. IF the number is negative, END the LOOP

    C. Add one to COUNT

    D. Add the number to SUM

4. Compute the average

5. Print the average

6. END the program

PAGE 85

BASIC Version   Flowchart Version

```
]LIST

10    REM   -- LOOPS - PROBLEM 3
20    REM   --
40    REM   -- THE PROBLEM READS IN, AND
42    REM   -- AVERAGES ALL NUMBERS UNTIL
44    REM   -- A NEGATIVE NUMBER APPEARS.
46    REM   --
80    DATA  933,233,876,239,347,110
85    DATA  45,129,930,3453,274,336
90    DATA  237,123,9,382,884,-84,456
109   REM
110   REM   -- INITIALIZATION OF LOOP
111   REM
120   LET SUM = 0
130   LET COUNT = 0
139   REM
140   REM   -- START OF LOOP
141   REM
150   READ NUMBER
159   REM
160   REM   -- CHECK FOR END OF LOOP
161   REM
170   IF NUMBER < 0 THEN 230
179   REM
180   REM   -- INCREMENT OF LOOP
181   REM
190   LET COUNT = COUNT + 1
200   LET SUM = SUM + NUMBER
209   REM
210   REM   -- REPEAT LOOP
211   REM
220   GOTO 140
229   REM
230   REM   -- END OF LOOP
231   REM
240   LET AVERAGE = SUM / COUNT
250   PRINT "THE AVERAGE IS ";AVERAGE
999   END

]RUN
THE AVERAGE IS 561.176471
```


PAGE 86
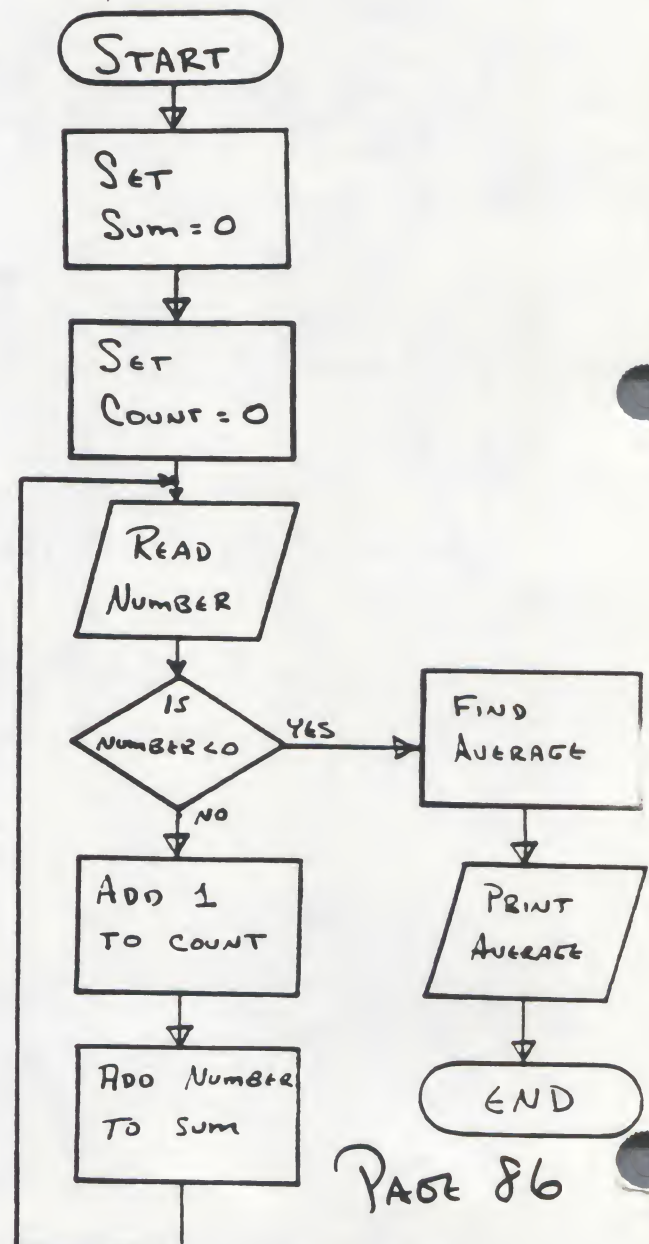
The program shows all of the steps of a loop. Each is there because the programmer must use each step to solve the problem. The FOR/NEXT version could not be used because we did not know how many times the loop needed to be executed.

If the sum were needed, but not the average, then steps 2, 3(A), and 4 could have been omitted from the Outline. Step 5 would have been to print the Sum. This would still have required each of the steps of the loop, but the program would have been shorter.

You will notice that there are 18 numbers read in, but only 17 are averaged. The last number read in is -84. Numbers after that are not read in.

The IF/THEN loop requires more steps than the FOR/NEXT loop. It is also generally more difficult to write correctly. Hence, whenever possible, the FOR/NEXT loop should be used. However, you should also realize that there will be times when you cannot use the FOR/NEXT loop.

SUMMARY OF STRUCTURES

The three structures used have been Sequences, Choices, and Loops. Some comments need to be made about the use of the three structures used.

1. There is always exactly one beginning place used in each.

2. There is always exactly one ending place used in each.

3. Each program can be described in outline form.

4. Some parts of the outline may need to be subdivided.

5. GOTO statements are used only within the Choice and Loop structures.

Programming needs to be planned. The three structures (Sequences, Choices, and Loops) help you to plan your programs. They also help you to explain your programs to other programmers. Knowing how to use these three structures effectively, you will find that programming new and more difficult programs will become more and more easy.

PAGE 87

Computer Literacy

Name:_____ Hour:_____ Date:_____

Sometimes it is very helpful to be able to count the number of times a
program goes through a loop. Later we will learn how to stop a program after
it has gone through a certain number of loops. But first we must find a way
to count these loops.

The first thing to do is pick a variable to use as the counter and make it
equal to 0. (C = 0)

So we can watch what is happening, we want to print C, then add 1 to C and
return to print the new C, add 1 to that, etc.

```
]LIST

100   LET C = 0
110   PRINT C
120   LET C = C + 1
130   GOTO 110
999   END
```

Type in this program and RUN it. It will be on an infinite loop so remember
to stop it press CTRL C.

You do not want line 130 to read 130 GOTO 100 because that would again bring
C back to 0 and you would only see 0 1 0 1 etc.

PAGE 88

Computer Literacy

SOUND AND LOW-RESOLUTION GRAPHICS

Graphics programs let you make pictures on the monitor screen with dots of light. Each dot which is really a small rectangle has an address or location of its own. Graphics programs tell the computer which rectangles to light up.

Graphics can be lots of fun because you can create pictures in color and also use sounds to liven the programs up. This is useful in creating computer games and visual images such as graphs, charts and messages, as well as, attention getters.

The idea behind graphics is that you want to create a picture. To create the picture, you must choose the color and tell the computer where to put a dot of that color on the screen. Sometimes you will put lines of color on the screen. This helps to create the picture even faster. So that we will know where we can put the color on the screen, we must know how the monitor screen is broken into pieces.

MAKING THE SCREEN INTO 1600 PARTS

The low resolution graphics screen is divided into 40 horizontal rows and 40 vertical columns. This actually gives you 1600 rectangular areas that can be filled with a dot. Look at the next page, called LOW RESOLUTION GRAPHICS WORKSHEET. Notice the numbers 0-39 going down the left side vertically. There are also numbers 0-39 along the top of the page.

To find a position on the screen, the PLOT statement can be used. The PLOT statement looks like PLOT X,Y. The X value is the horizontal position and the Y value is the vertical position. These two numbers will describe the exact rectangle to be colored on the screen. Both numbers must each be 0 or larger and 39 or smaller.

Remember the horizontal range is 0-39 from left to right (X coordinate) and the vertical reange is 0-39 from top to bottom (Y coordinate). For example, in PLOT 3,6 the 3 is the horizontal location (column 3) and the 6 is the vertical location (row 6). In mathematics we refer to the horizontal distance as the X-coordinate and the vertical distance as the Y-coordinate. So in PLOT 3,6 the X=3 and the Y=6. This is plotted on the partial graphics screen below.

PAGE 89

LOW RESOLUTION GRAPHICS WORKSHEET
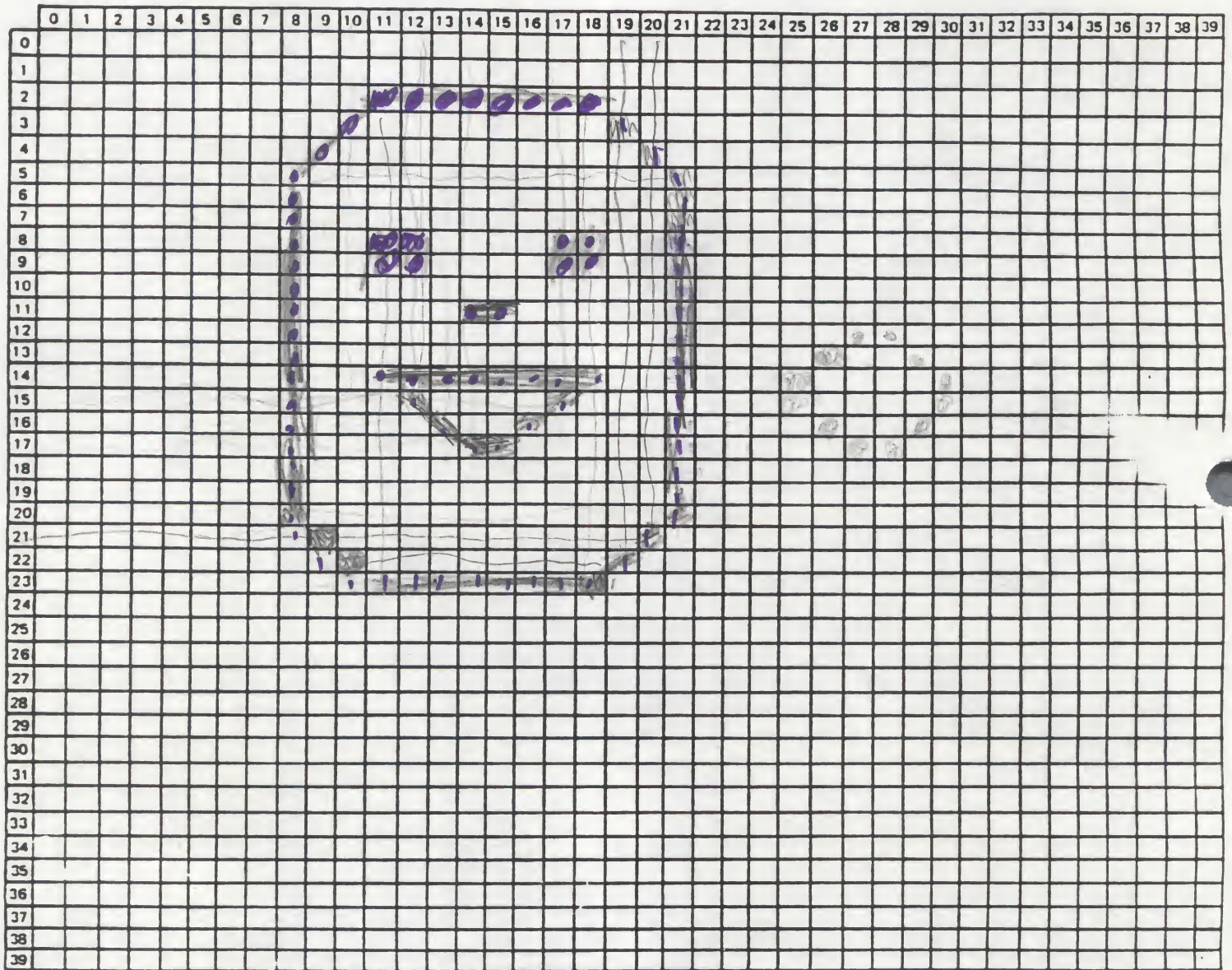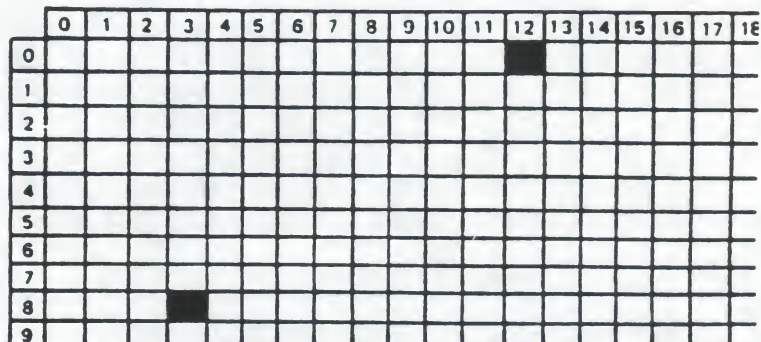
CREATING THE COLOR

In order to put a color on a screen, we must tell the computer which color to put on the screen. If you do not have a color monitor, then you will get various shades of grey. The Apple II Plus has sixteen colors (numbered 0-15) available. The list below shows you what number must be used to get a particular color. The two grey colors are the same.

0	BLACK	6	MEDIUM BLUE	11	PINK
1	MAGENTA (RED)	7	LIGHT BLUE	12	GREEN
2	DARK BLUE	8	BROWN	13	YELLOW
3	PURPLE	9	ORANGE	14	AQUA
4	DARK GREEN	10	GREY	15	WHITE
5	GREY				

The statement COLOR is used to tell the computer which color is currently being graphed on the monitor. The color being graphed will stay the same until it is changed. When Black (the zero color) is being added to the screen, it erases the other color that was there. The color black is added to the screen, but it often is not apparent to the person.

An example of plotting two points in orange is:

```
COLOR = 9
PLOT    3,8
PLOT    12,0
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0													■						
1																			
2																			
3																			
4																			
5																			
6																			
7																			
8				■															
9																			

GETTING IN AND OUT OF GRAPHICS

The computer must be told that graphics is going to be used. The statement GR (short for GRaphics) clears the top portion of the screen so that points, lines, and pictures can be drawn. It leaves four lines of space for textual material at the bottom of the screen. The statement TEXT takes the screen out of graphics mode and allows the entire screen to be used for textual material again.

IMMEDIATE AND PROGRAM MODE

You can type in individual instructions at the bottom of the screen and watch the picture be created as you go. This is called immediate mode. The disadvantage is that you cannot save the picture.

The other option is to write the graphics statements within a program. This offers many advantages, as you can now save it, you can use the many BASIC statements which you already know, and you will find it easier to edit and change.
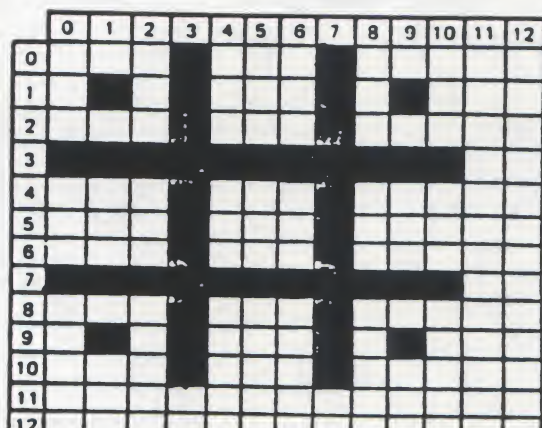
THE GRAPHICS STATEMENTS

We will now look at an example:

```
NEW
100    GR
110    COLOR = 2
120    HLIN  0,10  AT 3
130    HLIN  0,10  AT 7
140    VLIN  0,10  AT 3
150    VLIN 10, 0  AT 7
160    COLOR = 13
170    PLOT  1,1
180    PLOT  9, 1
190    PLOT  1, 9
200    PLOT  9, 9
999    END
RUN
TEXT
```

All of the different kinds of graphics statements are shown. Type in the example and watch what happens at each step.

SUMMARY OF GRAPHICS STATEMENTS

GR   By typing GR, all except the bottom 4 lines of the screen are put into graphics mode. The bottom four lines can still be used for text. That way you can see what you are typing.

TEXT   Typing TEXT cancels GR and thus changes it from graphics to text mode.

COLOR   This sets the screen color for future graphics. You can change the color by assigning a new number to color, depending upon the color you want. If you forget to set the color, everything will be plotted in COLOR=0 which is black and you won't see the picture although it is being plotted. In the example above, line 110 makes the color dark blue until it is changed to yellow in line 160.

PLOT   This allows you to put an already specified color in a specific location on the screen. Lines 170, 180, 190, and 200 above place a yellow dot in each of the outside parts of the tic-tac-toe board.

Another example is below:

```
NEW
100     GR
110     COLOR = 4
120     PLOT 3,5
130     PLOT 5,6
140     COLOR = 9
150     PLOT 8,7
160     PLOT 0,9
999     END
```

The plots in 120 and 130 will be dark green and those in 150 and 160 will be changed because the color was changed. You should be able to see a difference in the shades on your monitor. Sometimes the colors are not exactly as they should be because the monitor is not adjusted to exactly the color pattern needed.

HLIN   If you want to light up a whole line or part of a line across the screen, you would have to use many PLOT statements. However, there is an easier way.

The HLIN statement (short for Horizontal LINe) causes a horizontal line to be plotted at a particular row (vertical location). For example,

HLIN   3,31   AT   12

plots a line from position 3 through position 31 at horizontal line position 12. 29 continuous dots are shaded to form a line.

VLIN   The VLIN statement is the same as the HLIN statement except that it plots vertical lines. For example,

VLIN   0,39   AT   10   .

plots a line from the top position (position 0) through the bottom position (position 39) at Vertical position 10. Forty continuous dots are shaded to form a line.

CREATIVE SOUNDS

To generate sounds in a graphics program (or elsewhere), type

SOUND = PEEK(-16336)

To vary the pitch of the sound, you may combine the PEEK(-16336) with other PEEKs.

EXAMPLE: X=-16336
            SOUND = PEEK(X) - PEEK(X) - PEEK(X)

To make a sound occur at a given point in a program, combine an IF/THEN

statement with the sound generating statement.

EXAMPLE: 100 IF X>-1 AND X<40 THEN 150
           110 SOUND = PEEK(-16336)

This sequence makes a click whenever X has a value that is off the screen.

SOME THINGS TO REMEMBER ABOUT GRAPHICS

1. If you do your graphics in the immediate mode, you will not be able to SAVE and RUN your picture. However, you will see it as you are plotting it.

2. As always, if you want to SAVE a deferred program, you must have already "booted" the system with an initialized diskette.

3. If the screen is set for graphics, you will be able to see only the bottom four lines. Remember TEXT gets you back into the text mode.

4. When writing a graphics program be sure to put in the GR command.

5. If you don't state a color, everything will be black and you won't be able to see your drawing.

6. Remember that there are 40 spaces across and 40 spaces down but they are labeled 0-39. Using too large of numbers will give you an error message--?ILLEGAL QUANTITY ERROR.

7. You can generate sound in your graphics program by typing SOUND = PEEK(-16336) in the appropriate places.

Computer Literacy

SOUND & HIGH-RESOLUTION GRAPHICS

Graphics programs let you make pictures on the monitor screen with dots of light. Each dot which is really a tiny, tiny rectangle has an address or location of its own. Graphics programs tell the computer which dots to light up.

Graphics can be lots of fun because you can create pictures in color and also use sounds to liven the programs up. This is useful in creating computer games and visual images such as graphs, charts and messages, as well as, attention getters.

The idea behind graphics is that you want to create a picture. To create the picture, you must choose the color and tell the computer where to put a dot of that color on the screen. Sometimes you will put lines of color on the screen. This helps to create the picture even faster. So that we will know where we can put the color on the screen, we must know how the monitor screen is divided into pieces.

MAKING THE SCREEN INTO 44,800 PARTS

The high resolution graphics screen is divided into 160 horizontal rows and 280 vertical columns. That actually gives you 44,800 points or dots which can be placed on the monitor. Look at the next page which is called HIGH RESOLUTION GRAPHICS WORKSHEET. Notice the numbers 0-279 going across the top from left to right horizontally. Also, notice the numbers 0-159 going down the left side vertically.

To find a position on the screen, the HPLOT statement can be used. The HPLOT statement looks like HPLOT X,Y. The X value is the horizontal position and the Y value is the vertical position. These two numbers will describe the exact dot to be colored on the screen. The first number (the X coordinate) must be between 0 and 279. The second number (the Y coordinate) must be between 0 and 159.

Remember the horizontal range is 0-279 from left to right and the vertical range is from 0 to 159 from top to bottom.

For example, in HPLOT 30,59 the 30 is the horizontal location (column 30) and the 59 is the vertical position (row 59). In mathematics we refer to the horizontal as the X-coordinate and the vertical as the Y-coordinate.

CREATING THE COLOR

In order to put a color on a screen, we must tell the computer which color to put on the screen. If you do not have a color monitor, then you will get various shades of grey. The Apple II Plus has 8 colors available (numbered 0-7). Several of these are the same colors. The list below shows you what number must be used to get a particular color.

PAGE 95

HIGH RESOLUTION GRAPHICS WORKSHEET

PAGE 96
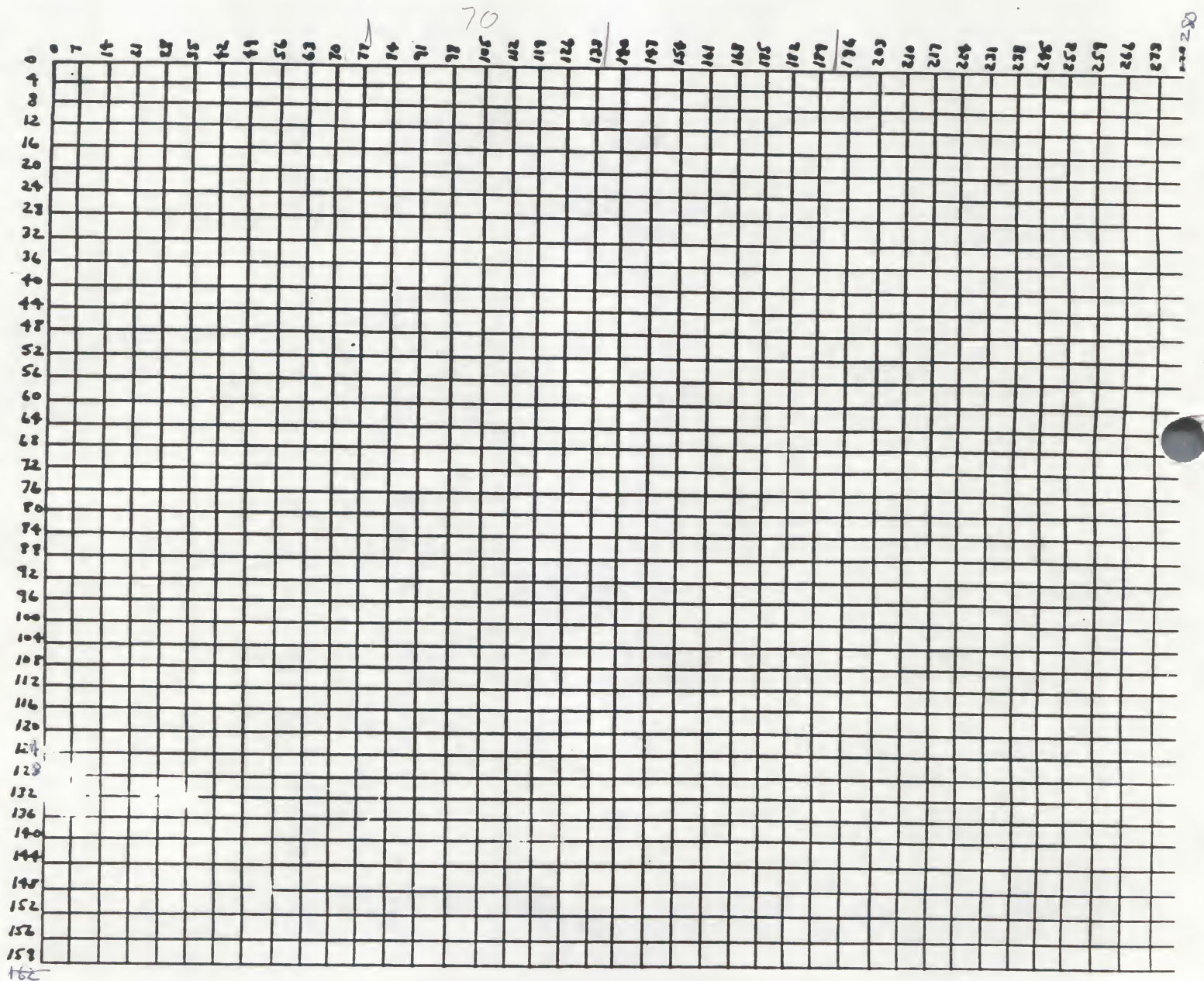
0	BLACK1	4	BLACK2
1	GREEN	5	ORANGE
2	VIOLET	6	BLUE
3	WHITE1	7	WHITE2

The statement HCOLOR is used to tell the computer which color is currently being graphed on the monitor. The color being graphed will stay the same until it is changed. When Black (the zero and four colors) is being added to the screen, it erases the other color that was there. The color black is added to the screen, but it often is not apparent to the person.

An example of plotting two blue points is:

```
HCOLOR =    6
HPLOT   100,5
HPLOT   200,5
```

GETTING IN AND OUT OF GRAPHICS

The computer must be told that graphics is going to be used. The statement HGR (short for High resolution GRaphics) clears the top portion of the screen so that points, lines, and pictures can be drawn. It leaves four lines of space for textual material at the bottom of the screen. The statement TEXT takes the screen out of graphics mode and allows the entire screen to be used for textual material again.

IMMEDIATE AND PROGRAM MODE

You can type in individual instructions and statements at the bottom of the screen and watch the picture be created as you go. The disadvantage is that you cannot save the picture.

The other option is to write the graphics statements within a program. This offers many advantages, as you can now save it, you can use the many BASIC statements which you already know, and you will find it easier to edit and change.

THE GRAPHICS STATEMENTS

We will now look at an example:

```
NEW
100     HGR
110     HCOLOR = 1
120     HPLOT    100,10 TO 200,10
130     HPLOT    TO 250,70
140     HPLOT    TO 200,130
150     HPLOT    TO 100,130
160     HPLOT    TO  50,70
170     HPLOT    TO 100,10
180     HCOLOR = 5
```

```
190      HPLOT  101,51
200      HPLOT  201,51
210      HPLOT  100,100 TO 200,100
999      END
RUN
TEXT
```

All of the different kinds of high-resolution statements are shown. Type in the example and watch what happens at each step. You should get a green face with orange eyes and mouth.

SUMMARY OF GRAPHICS STATEMENTS

HGR   By typing HGR, all except the bottom 4 lines of the screen are put into graphics mode. The bottom four lines can still be used for text, so that you can see what you are typing.

TEXT   Typing TEXT cancels HGR and thus changes it from graphics to text mode.

HCOLOR This sets the color screen for future graphics. You can change the color by assigning a new number to HCOLOR, depending upon the color you want. If you forget to set the color, everything will be plotted in HCOLOR = 0 which is black and you won't see the picture although it is being plotted. In the example above, line 110 makes the color green until it is changed to orange in line 180.

HPLOT   This allows you to put an already specified color in a specific location on the screen. It can be used three separate ways.

        A. You can plot a single point. Lines 190 and 200
            in the program above are two examples.

        B. You can plot a line. The line can be
            horizontal, vertical, or diagonal.
            Lines 120 and 210 are two examples in
            the program above.

        C. This is a second way to draw a line
            from a previous point.
            Simply indicate the point to which
            you want it drawn. Lines 130,140,150
            160, and 170 are examples. This allows
            you to continue the line, but change
            the direction.

CREATING SOUNDS

     To generate sounds in a graphics program (or elsewhere), type

PAGE 98

SOUND = PEEK(-16336)

To vary the pitch of the sound, you may combine the PEEK(-16336) with other PEEKs.

EXAMPLE: X=-16336
           SOUND = PEEK(X) - PEEK(X) - PEEK(X)

To make a sound occur at a given point in a program, combine an IF/THEN statement with the sound generating statement.

EXAMPLE: 100 IF X>-1 AND X<280 THEN 150
           110 SOUND = PEEK(-16336)

This sequence makes a click whenever X has a value that is off the screen.

SOME THINGS TO REMEMBER ABOUT GRAPHICS

1. If you do your graphics in the immediate mode, you will not be able to SAVE and RUN your picture. However, you will see it as you are plotting.

2. As always, if you want to SAVE a deferred program, you must have already "booted" the system with an initialized diskette.

3. If the screen is set for graphics, you will be able to see only the bottom four lines. Remember TEXT gets you back into the text mode.

4. When writing a graphics program, be sure to put in the HGR statement.

5. If you don't state a color, everything will be black and you won't be able to see your drawing.

6. Remember that there are 280 spaces across and 160 spaces down but they are labeled 0-279 and 0-159. Using too large of numbers will give you an error message--?ILLEGAL QUANTITY ERROR.

7. You can generate sound in your graphics program by typing SOUND = PEEK(-16336) in the appropriate places.

Computer Literacy

<u>BASIC STATEMENT FORMATS</u>

Note: Parentheses indicate that it is optional.

REM

line # REM comments

Example: 100 REM -- WRITTEN BY JOHN SMITH

LET

line # (LET) variable = expression

Examples: 100 LET A = B + C
           110 A = B + C

PRINT

line # PRINT expression

Examples: 100 PRINT
           110 PRINT A
           120 PRINT "THE ANSWER IS", A
           130 PRINT A; B; C

END

line # END

Example: 999 END

INPUT

line # INPUT (expression in quotes;) variables

Examples: 100 INPUT A, B, C
           110 INPUT "ENTER TWO NUMBERS"; A, B

READ

line # READ variables

Examples: 100 READ A, B, C

DATA

line # DATA values

Examples: 90 DATA 4, 3, "NEW YORK", 8, 7

GOTO

line # GOTO line #

Examples: 100 GOTO 200

IF/THEN

line # IF relation THEN line #
line # IF relation THEN statement

Examples: 100 IF A = B THEN 200
110 IF A = C + D THEN READ A,B

FOR

line # FOR variable = number or variable TO number or variable
(STEP number or variable)

Examples: 100 FOR I = 1 TO 200
110 FOR J = C TO 150
120 FOR K = -7 TO 9 STEP 2

NEXT

line # NEXT variable

Examples: 100 NEXT I

\

PAGE 101

Computer Literacy

<div align="center">SUMMARY 1</div>

CURSOR - The name of the flashing square which tells you that the computer
is ready to accept your input.

HOME - clears the screen.

LET - This statement assigns a value to a variable. The word LET is
optional.
Example: LET X = 15 or X = 15

PRINT - This statement tells the computer to print the work you have done.
You may print information in quotes. You may also print a simple
number or the result of an arithmetic operation.

Example: PRINT "DON'T FORGET THE QUOTES"

PRINT 3

PRINT 3 + 7

PROMPT -   - This tells you that the computer is in Applesoft BASIC.
Other prompts are > (Integer BASIC) and * (machine language).

RESERVED WORDS - These are special words which the computer understands.
These cannot be used as variable names or as a part of variable
names.

RETURN - This key must be pressed whenever you have finished a line. This
tells the computer that you have finished that line.

SYNTAX ERROR - The computer gives you this message when it does not under-
stand what you have typed.

VARIABLES - These are names used to store information in the computer. There
are two types of variables: numeric and string.
Numeric variables can only be given values which are numbers.
They can be used in calculations. String variables
must end with $. Their value must be enclosed in quotes.

All variables must start with a letter, but can contain letters
and numbers.

SUMMARY 2

SCIENTIFIC NOTATION - a method of writing very large or very small numbers. In the computer, it is used with large numbers of more than nine digits, or small numbers closer to zero than 0.01.

The format is:

A non-zero digit
A decimal point
All other digits except for trailing zeroes
The letter E (it means Exponent)
A plus or minus sign
A two-digit exponent

Examples:

1250000000000 = 1.25E+12 (1.25×10^{12})

0.00342 = 3.42E-03 (3.42×10^{-3})

ORDER OF OPERATION - All calculations are done from left to right using the following order:

1) Work within parenthesis

$4 * (3 + 7)$ (Answer is 40)

2) Exponents   (denoted by   )

2^3 becomes 2 ∧ 3 (Answer is 8)

3) Multiplication (*) and division (/) as you come to it

3*4/6 (Answer is 2)

6/2*3 (Answer is 9)

6*2/3 (Answer is 4)

4) Addition (+) and subtraction (-) as you come to them.

5+4-3 (Answer is 6)

5-4+3 (Answer is 4)

Example:

    3 * [4 + (5 - 2)] + 36/3 ∧ 2 * 8

    becomes

    3 * (4 + (5 - 2)) + 36 / 3∧2 * 8

    3 * (4 + 3) + 36 / 3∧2 * 8

    3 *   7   + 36 / 3∧2 * 8

    3 *   7   + 36 / 9   * 8

      21   +   4   * 8

      21   +   32

             53

Computer Literacy

A <u>PROGRAM</u> is a set of instructions to the computer which tells the computer how to solve a problem.

Each statement in the program is identified by a LINE NUMBER. The computer will execute the program in order from the lowest numbered line number to the highest.

The order in which you type in the line numbers does not matter to the computer, as it will rearrange them in order. However, to make sense to the typist, they should be done in order.

A program should always have an END, making this line 999 or 9999. It should be the last statement in a program.

Do not number the lines in sequence, but skip, generally, every ten. This allows you to add lines later if necessary.

To add a statement, give it a line number which is between the line numbers of the statements which you wish to make an insertion.

To erase a line, type the line number and press RETURN.

LIST tells the computer to show you the statements of the program.

RUN tells the computer to follow the instructions, or execute the program.

PR#6 turns on the disk drive and "boots the disk."

CATALOG is a DOS (Disk Operating System) command which results in the listing of the names of all the programs on a diskette.

SAVE (name) is a DOS (Disk Operating System) command which saves the program from the computer's memory to the diskette under the (name) used.

LOAD (name) is a DOS command which loads a program from the diskette into the computer's memory.

RUN (name) is a DOS command which loads a program from the diskette into the computer's memory <u>and</u> causes that program to be run.

NEW erases a program from the computer's memory.

COMMAS in PRINT statements cause columns to be printed.

SEMICOLONS cause the information to be printed immediately next to each other.

REM is a command meaning REMARK. It is ignored by the computer but can give information to someone looking at the listing of a program.

Delete tells the computer to erase a program on a disk

LOW-RESOLUTION GRAPHICS STATEMENT SUMMARY

FUNCTION		EXAMPLES
GR	Converts screen to graphics mode (GR means graphics)	GR
COLOR	Sets the color for plotting in low-resolution graphics mode (there are 16 colors available numbered 0-15).	COLOR=15(white) COLOR= 1(red) COLOR=12(green)
PLOT X,Y	Places a dot at the location specified by the X,Y coordinates. PLOT allows you to turn on or light up a spot at location X,Y. The color of the spot is determined by the most recent value of COLOR, which is 0(black) if not specified otherwise. X and Y values range from 0-39.	PLOT 0,0 puts a dot in the upper left corner PLOT 0,39 (dot in the lower left) PLOT 39,0 (dot in the upper right PLOT 39,39 (dot in the lower right)
HLIN	Short for horizontal line, causes a horizontal line to be plotted at a particular row.	HLIN 2,8 at 6 gives you a horizontal line in columns 2 thru 8 in row 6.
VLIN	Short for vertical line, causes a vertical line to be plotted in a particular column.	VLIN 2,8 at 6 gives you a vertical line in rows 2 thru 8 in column 6.
TEXT	Converts the entire screen back to TEXT (words) mode. Remember TEXT mode can display up to 40 characters per line and up to 24 lines.	TEXT

PAGE 106

HIGH-RESOLUTION GRAPHICS STATEMENT SUMMARY

	FUNCTION	EXAMPLES
HGR	Converts screen to graphics mode (HGR means graphics)	HGR
HCOLOR	Sets the color for plotting in high-resolution graphics mode (there are 8 colors available numbered 0-7).	HCOLOR= 5(orange) HCOLOR= 1(green) HCOLOR= 3(white)
HPLOT X,Y	Places a dot at the location specified by the X,Y coordinates. HPLOT allows you to turn on or light up a spot at location X,Y.	HPLOT 0,0 puts a dot in the upper left corner HPLOT 0,159 (dot in the lower left)
	The color of the spot is determined by the most recent value of HCOLOR, which is 0(black) if not specified otherwise.	HPLOT 279,0 (dot in the upper right) HPLOT 279,159 dot in the lower right)
	X values range from 0-279.	
	Y values range from 0-159.	
HPLOT X1,Y1 TO X2,Y2	Causes a line to be created from X1,Y1 to X2,Y2. The line can be horizontal, vertical, or diagonal.	HPLOT 0,0 TO 279,159 (line from upper left-hand corner to lower right-hand corner)
HPLOT TO X3,Y3	Causes a line to be created from wherever it was to X3,Y3. The line can be horizontal, vertical or diagonal.	HPLOT TO 10,50 (line is drawn from wherever it was to point 10,50)
TEXT	Converts the entire screen back to TEXT (words) mode. Remember TEXT mode can display up to 40 characters per line and up to 24 lines.	TEXT

PAGE 107

RESERVED WORDS

Some names are not allowed for variables because they include a word that has a special meaning to the Apple. These are RESERVED words. One of these words which you may have used is HOME. Thus, the name of a variable must not have HOME in it.

Try typing

    HOMEBOUND = 9

or

    ATHOME = 5

Did you get SYNTAX ERROR?

Whenever a variable name gives you the ?SYNTAX ERROR message, it means that you have mistyped the line in some way or you have included a reserved word in the name. Choose another name.

Following is a list of the RESERVED words.

&	GET	NEW	SAVE
	GOSUB	NEXT	SCALE=
ABS	GOTO	NORMAL	SCRN(
AND	GR	NOT	SGN
ASC		NOTRACE	SHLOAD
AT	HCOLOR=		SIN
ATN	HGR	ON	SPC(
	HGR2	ONERR	SPEED=
CALL	HIMEN:	OR	SQR
CHR$	HLIN		STEP
CLEAR	HOME	PDL	STOP
COLOR=	HPLOT	PEEK	STORE
CONT	HTAB	PLOT	STR$
COS		POKE	
	IF	POP	TAB(
DATA	IN#	POS	TAN
DEF	INPUT	PRINT	TEXT
DEL	INT	PR#	THEN
DIM	INVERSE		TO
DRAW		READ	TRACE
	LEFT$	RECALL	
END	LEN	REM	USR
EXP	LET	RESTORE	
	LIST	RESUME	VAL
FLASH	LOAD	RETURN	VLIN
FN	LOG	RIGHT$	VTAB
FOR	LOMEM:	RND	
FRE		ROT=	WAIT
	MID$	RUN	
			XPLOT
CL/9-82/a			XDRAW

Page 108